Sound and Complete Verification of Polynomial Networks

Elias Abad Rocamora[*1] **Mehmet Fatih Sahin**[2] **Fanghui Liu**[2] **Grigorios Chrysos**[2] **Volkan Cevher**[2]

Abstract

Polynomial Networks (PNs) have demonstrated promising performance on face and image recognition recently. However, robustness of PNs is unclear and thus obtaining certificates becomes imperative for enabling their adoption in real-world applications. Existing verification algorithms on ReLU neural networks (NNs) based on branch and bound (BaB) techniques cannot be trivially applied to PN verification. In this work, we devise a new bounding method, equipped with BaB for global convergence guarantees, called VPN. One key insight is that we obtain much tighter bounds than the interval bound propagation baseline. This enables sound and complete PN verification with empirical validation on MNIST, CIFAR10 and STL10 datasets. We believe our method has its own interest to NN verification.

1. Introduction

Polynomial Networks (PNs) have demonstrated promising performance across image recognition and generation (Chrysos et al., 2021b; Chrysos & Panagakis, 2020) being state-of-the-art on large-scale face recognition[†].Unlike the conventional Neural Networks (NNs), where nonlinearitiy is introduced with the use of activation functions (LeCun et al., 2015), PNs learn non-linear mappings without the need of activation functions by exploiting multiplicative interactions (Hadamard products). Recent works have uncovered interesting properties of PNs, like their larger model expressivity (Fan et al., 2021) or their spectral bias (Choraria et al., 2022). However, one critical issue before considering PNs for real-world applications is their robustness.

Neural networks are prone to small (often imperceptible to the human eye), but malicious perturbations in the input data points (Szegedy et al., 2014; Goodfellow et al., 2015). Those perturbations can have a detrimental effect on image recognition systems, e.g., as illustrated in face recognition (Goswami et al., 2019; Zhong & Deng, 2019; Dong et al., 2019; Li et al., 2020). Guarding against such attacks has so far proven futile (Shafahi et al., 2019; Dou et al., 2018).Instead, a flurry of research has been published on certifying robustness of NNs against this performance degradation (Katz et al., 2017; Ehlers, 2017; Tjeng et al., 2019; Bunel et al., 2020a; Wang et al., 2021; Ferrari et al., 2022). However, most of the verification algorithms for NNs are developed for the ReLU activation function by exploiting its piecewise linearity property and might not trivially extend to other nonlinear activation functions (Wang et al., 2021). Indeed, Zhu et al. (2022) illustrate that guarding PNs against adversarial attacks is challenging. Therefore, we pose the following question: *Can we obtain certifiable performance for PNs against adversarial attacks?*

In this work, we answer affirmatively and provide a method for the verification of PNs. Concretely, we take advantage of the twice-differentiable nature of PNs to build a lower bounding method based on α-convexification (Adjiman & Floudas, 1996), which is integrated into a Branch and Bound algorithm (Land & Doig, 1960) to guarantee completeness of our verification method. In order to use α-convexification a lower bound α of the minimum eigenvalue of the Hessian matrix over the possible perturbation set is needed. We use interval bound propagation together with the theoretical properties of the lower bounding Hessian matrix (Adjiman et al., 1998), in order to develop an algorithm to efficiently compute α.

Our *contributions* can be summarized as follows: (i) We propose the first algorithm for the verification of PNs. (ii) We thoroughly analyze the performance of our method by comparing it with a black-box solver and an interval bound propagation (IBP) BaB algorithm. (iii) We empirically show that using α-convexitication for lower bounding provides tighter bounds than IBP for PN verification.

[1]UPC, Barcelona, Spain [2]LIONS, EPFL, Lausanne, Switzerland. Correspondence to: Elias Abad Rocamora <abad.elias00@gmail.com>.

1st Workshop on Formal Verification of Machine Learning, Baltimore, Maryland, USA. Colocated with ICML 2022.

[*]Work developed at LIONS, EPFL

[†]https://paperswithcode.com/sota/face-verification-on-megaface

The proposed approach can practically verify PNs and that could theoretically be applied for sound and complete verification of any twice-differentiable network.

Notation: We use the shorthand $[n] := \{1, 2, \dots, n\}$ for a positive integer n. We use bold capital (lowercase) letters, e.g., $\boldsymbol{X}$ ($\boldsymbol{x}$) for representing matrices (vectors). The j^{th} column of a matrix $\boldsymbol{X}$ is given by $\boldsymbol{x}_{:j}$. The element in the i^{th} row and j^{th} column is given by x_{ij}, similarly, the i^{th} element of a vector $\boldsymbol{x}$ is given by x_i. The element-wise (Hadamard) product, symbolized with $*$, of two matrices (or vectors) in $\mathbb{R}^{d_1 \times d_2}$ (or $\mathbb{R}^d$) gives another matrix (or vector) in $\mathbb{R}^{d_1 \times d_2}$ (or $\mathbb{R}^d$). The ℓ_∞ norm of a vector $\boldsymbol{x} \in \mathbb{R}^d$ is given by: $||\boldsymbol{x}||_\infty = \max_{i \in [d]} |x_i|$. Lastly, the operators $\mathcal{L}$ and $\mathcal{U}$ give the lower and upper bounds of a scalar, vector or matrix function by IBP, see Section 3.1.

2. Background

To make the paper self-contained, we introduce the PN architecture in Section 2.1 and the Robustness Verification problem in Section 2.2.

2.1. Polynomial Networks (PNs)

Polynomial Networks (PNs) are inspired by the fact that any smooth function can be approximated via a polynomial expansion (Stone, 1948). However, the number of parameters increases exponentially with the polynomial degree, which makes it intractable to use high degree polynomials for high-dimensional data problems such as image classification where the input can be in the order of 10^5 (Deng et al., 2009). Chrysos et al. (2021b) introduce a joint factorization of polynomial coefficients in a low-rank manner, reducing the number of parameters to linear with the polynomial degree and allowing the expression as a Neural Network. We briefly recap one fundamental factorization below.

Let N be the polynomial degree, $\boldsymbol{z} \in \mathbb{R}^d$ be the input vector, d, k and o be the input, hidden and output sizes, respectively. The recursive equation of PNs can be expressed as:

$$\boldsymbol{x}^{(n)} = (\boldsymbol{W}_{[n]}^\top \boldsymbol{z}) * \boldsymbol{x}^{(n-1)} + \boldsymbol{x}^{(n-1)}, \forall\, n \in [N], \quad (1)$$

where $\boldsymbol{x}^{(1)} = \boldsymbol{W}_{[1]}^\top \boldsymbol{z}$, $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{x}^{(N)} + \boldsymbol{\beta}$ and $*$ denotes the Hadamard product. $\boldsymbol{W}_{[n]} \in \mathbb{R}^{d \times k}$ and $\boldsymbol{C} \in \mathbb{R}^{o \times k}$ are weight matrices, $\boldsymbol{\beta} \in \mathbb{R}^o$ is a bias vector. A graphical representation of a third degree PN architecture corresponding to Eq. (1) can be found in Fig. 3. Further details on the factorization (as well as other factorizations) are deferred to the Appendix B.1 (Appendix B.2).

2.2. Robustness Verification

Robustness Verification (Bastani et al., 2016; Liu et al., 2021) consists of verifying that a property regarding the input and output of a NN is satisfied, e.g. checking whether or not a small perturbation in the input will produce a change in the network output that makes it classify the input into another class. Let $f : [0, 1]^d \to \mathbb{R}^o$ be a function, e.g., a NN or a PN, that classifies inputs $\boldsymbol{z}$ into a class c, such that $c = \arg\max \boldsymbol{f}(\boldsymbol{z})$. We want to verify that for any input satisfying a set of constraints C_{in}, the output of the network will satisfy a set of output constraints C_{out}. That is, we want to check the following logical formula is satisfied:

$$\boldsymbol{z} \in C_{\text{in}} \implies \boldsymbol{f}(\boldsymbol{x}) \in C_{\text{out}}. \quad (2)$$

In this work we focus on *adversarial robustness* (Szegedy et al., 2014; Carlini & Wagner, 2017) in classification. Assume an observation $\boldsymbol{z}_0$ is given and let $t = \arg\max \boldsymbol{f}(\boldsymbol{z}_0)$ be the correct class, we want to check whether every input in a neighbourhood of $\boldsymbol{z}_0$, is classified as t. We focus on adversarial attacks restricted to neighbourhoods defined in terms of ℓ_∞ norm, which is a popular norm-bounded attack in the verification community. Then, the constraint sets become:

$$\begin{aligned} C_{\text{in}} &= \{\boldsymbol{z} : ||\boldsymbol{z} - \boldsymbol{z}_0||_\infty \leq \epsilon, z_i \in [0, 1], \forall i \in [d]\} \\ &= \{\boldsymbol{z} : \max\{0, z_{0i} - \epsilon\} \leq z_i \\ &\qquad \leq \min\{1, z_{0i} + \epsilon\}, \forall i \in [d]\} \\ C_{\text{out}} &= \{\boldsymbol{y} : y_t > y_j, \forall j \neq t\}. \end{aligned} \quad (3)$$

This can be reformulated as a mathematical optimization problem. For every adversarial class $\gamma \neq t = \arg\max \boldsymbol{f}(\boldsymbol{z}_0)$, we can solve:

$$\min_{\boldsymbol{z}} \quad g(\boldsymbol{z}) = f(\boldsymbol{z})_t - f(\boldsymbol{z})_\gamma \quad \text{s.t.} \quad \boldsymbol{z} \in \mathcal{C}_{\text{in}}\,. \quad (4)$$

If the solution $\boldsymbol{z}^*$ with $v^* = f(\boldsymbol{z}^*)_t - f(\boldsymbol{z}^*)_\gamma \leq f(\boldsymbol{z})_t - f(\boldsymbol{z})_\gamma, \forall \boldsymbol{z} \in \mathcal{C}_{\text{in}}$ satisfies $v^* > 0$ then robustness is verified for the adversarial class γ.

3. Method

Our method, called VPN, can be categorized in the the Branch and Bound (BaB) framework (Land & Doig, 1960), a well known approach to global optimization (Horst & Tuy, 1996) and NN verification (Bunel et al., 2020a). This kind of algorithms guarantee finding a global minima of the problem in Eq. (4) by recursively splitting the original feasible set into disjoint sets (branching) where upper and lower bounds of the global minima are computed (bounding). This mechanism can be used to discard subsets where the global minima cannot be achieved (its lower bound is greater than the upper bound of another subset).

Our method is based on an α-BaB algorithm (Adjiman et al., 1998), which is characterized for using α-convexification (Adjiman & Floudas, 1996) for computing a lower bound of the global minima of each subset. To be specific, α-convexification aims to obtain a convex lower bounding

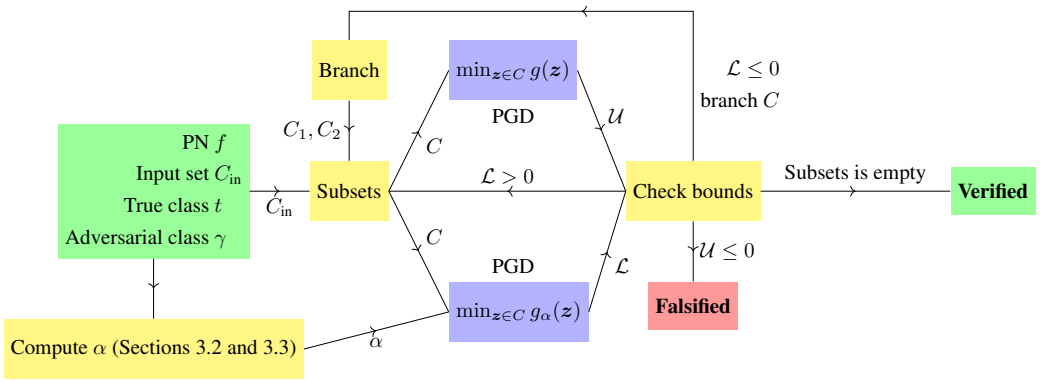

Figure 1: Overview of our branch and bound verification algorithm. Given a trained PN f, an input set C_{in}, the true class t and an adversarial class γ, we check if an adversarial example exists (**Falsified**) or not (**Verified**). Note that the branching of a subset C provides two disjoint subsets C_1 and C_2. Also note that when $\mathcal{L} > 0$, no subset is added to subsets.

function of any twice-differentiable function $g : \mathbb{R}^d \to \mathbb{R}$ such that

$$g_\alpha(\boldsymbol{z}; \alpha, \boldsymbol{l}, \boldsymbol{u}) = g(\boldsymbol{z}) + \alpha \sum_{i=1}^{d} (z_i - l_i)(z_i - u_i)\,, \quad (5)$$

as its α-convexified version. Let $\boldsymbol{H}_g(\boldsymbol{z}) = \nabla^2_{\boldsymbol{zz}} g(\boldsymbol{z})$ be the Hessian matrix of f, g_α is convex in $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$ for $\alpha \geq \max\{0, -\frac{1}{2}\min\{\lambda_{\min}(\boldsymbol{H}_g(\boldsymbol{z})) : \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\}\}$. Moreover, it holds that $g_\alpha(\boldsymbol{z}; \alpha, \boldsymbol{l}, \boldsymbol{u}) \leq g(\boldsymbol{z}), \forall \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$.

To make PN verification feasible, we need to study IBP for PNs and design an efficient estimate on the α parameter, which are our main technical contributions in the algorithmic aspect. In our case, every feasible set, starting with the input set $\mathcal{C}_{\text{in}}$ (Eq. (3)), is split by taking the widest variable interval and dividing it in two by the middle point. Then, the upper bound of each subproblem is given by applying standard Projected Gradient Descent (PGD) (Kelley, 1999) over the original objective function. This is a common approach to find adversarial examples (Madry et al., 2018), but as the objective is non-convex, it is not sufficient for sound and complete verification. The lower bound is given by applying PGD over the α-convexified objective g_α, as it is convex, PGD converges to the global minima and a lower bound of the original objective. The α parameter is computed only once per verification problem. Further details on the algorithm and the proof of convergence of Eq. (4) exist in Appendix C. See Fig. 1 for an overview of our method.

In the sequel, we detail our method to compute a lower bound on the minimum eigenvalue of the Hessian matrix into three main components: interval propagation, lower bounding Hessians, and fast estimation on such lower bounding via power method.

3.1. Interval Bound Propagation through a PN

Interval bound propagation (IBP) is a key ingredient of our verification algorithm. Suppose we have an input set defined by an ℓ_∞-norm ball like in Eq. (3). This set can be represented as a vector of intervals $\tilde{\boldsymbol{z}} = ([l_1, u_1]^\top, [l_2, u_2]^\top, \cdots, [l_d, u_d]^\top) = [\boldsymbol{l}, \boldsymbol{u}] \in \mathbb{R}^{d \times 2}$, where $[l_i, u_i]$ are the lower and upper bound for the i^{th} coordinate. Let $\mathcal{L}$ and $\mathcal{U}$ be the lower and upper bound IBP operators. Given this input set, we would like to obtain bounds on the output of the network $(f(\boldsymbol{z})_i)$, the gradient $(\nabla_{\boldsymbol{z}} f(\boldsymbol{z})_i)$, and the Hessian $(\nabla^2_{\boldsymbol{zz}} f(\boldsymbol{z})_i)$ for any $\boldsymbol{z} \in \tilde{\boldsymbol{z}}$. The operators $\mathcal{L}(g(\boldsymbol{z}))$ and $\mathcal{U}(g(\boldsymbol{z}))$ of any function $g : \mathbb{R}^d \to \mathbb{R}$ satisfy $\mathcal{L}(g(\boldsymbol{z})) \leq g(\boldsymbol{z}) \leq \mathcal{U}(g(\boldsymbol{z})), \forall \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$. We will define these upper and lower bound operators in terms of the operations present in a PN.

Using interval propagation (Moore et al., 2009), we can define:

$$\textbf{Identity} \; \{\mathcal{L}(z_i) = l_i, \quad \mathcal{U}(z_i) = u_i$$

$$\textbf{linear mapping} \begin{cases} \mathcal{L}(\sum_i w_i h_i(\boldsymbol{z})) = \sum_i w_i^+ \mathcal{L}(h_i(\boldsymbol{z})) \\ \qquad\qquad + w_i^- \mathcal{U}(h_i(\boldsymbol{z})) \\ \mathcal{U}(\sum_i w_i h_i(\boldsymbol{z})) = \sum_i w_i^- \mathcal{L}(h_i(\boldsymbol{z})) \\ \qquad\qquad + w_i^+ \mathcal{U}(h_i(\boldsymbol{z})) \end{cases}$$

$$\textbf{multiplication} \begin{cases} S = \begin{Bmatrix} \mathcal{L}(h_1(\boldsymbol{z}))\,\mathcal{L}(h_2(\boldsymbol{z})), \\ \mathcal{L}(h_1(\boldsymbol{z}))\,\mathcal{U}(h_2(\boldsymbol{z})), \\ \mathcal{U}(h_1(\boldsymbol{z}))\,\mathcal{L}(h_2(\boldsymbol{z})), \\ \mathcal{U}(h_1(\boldsymbol{z}))\,\mathcal{U}(h_2(\boldsymbol{z})) \end{Bmatrix}, |S| = 4 \\ \mathcal{L}(h_1(\boldsymbol{z})h_2(\boldsymbol{z})) = \min S\,, \\ \mathcal{U}(h_1(\boldsymbol{z})h_2(\boldsymbol{z})) = \max S\,, \end{cases} \quad (6)$$

where $w_i^+ = \max\{0, w_i\}$ and $w_i^- = \min\{0, w_i\}$, $h_i(\boldsymbol{z})$ is

any real-valued function of $\boldsymbol{z}$ and $|\cdot|$ is the set cardinality.

With these basic operations one can define bounds on any intermediate output, gradient or Hessian of a PN, for more details, we refer to Appendix B. which only consists on a linear mapping and a multiplication of intervals. We extend the upper and lower bound ($\mathcal{L}(\cdot)$ and $\mathcal{U}(\cdot)$) operators to also work on vectors and matrices by applying them at every position of the vector or matrix. One can directly use IBP to obtain bounds on the verification objective from Eq. (4) with a single forward pass of the bounds through the network and obtaining $\mathcal{L}(g(\boldsymbol{z})) = \mathcal{L}(f(\boldsymbol{z})_t) - \mathcal{U}(f(\boldsymbol{z})_\gamma)$. IBP is a common practice in NN verification to obtain fast bounds (Wang et al., 2018a).

3.2. Lower bound of the minimum eigenvalue of the Hessian matrix

Here we describe our method to compute a lower bound on the minimum eigenvalue of the Hessian matrix in the feasible set. Before deriving the lower bound, we need the first and second order partial derivatives of PNs.

Let $g(\boldsymbol{z}) = f(\boldsymbol{z})_t - f(\boldsymbol{z})_a$ be the objective function for $t = \arg\max \boldsymbol{f}(\boldsymbol{z}_0)$ and any $a \neq t$. In order to compute the parameter α for performing α-convexification, we need to know the structure of our objective function. In this section we compute the first and second order partial derivatives of the PN. The gradient and Hessian matrices of the objective function (see Eq. (4)) are easily found to be:

$$\begin{aligned} \nabla_{\boldsymbol{z}} g(\boldsymbol{z}) &= \sum_{i=1}^{k} (c_{ti} - c_{\gamma i}) \nabla_{\boldsymbol{z}} x_i^{(N)} \\ \boldsymbol{H}_g(\boldsymbol{z}) &= \sum_{i=1}^{k} (c_{ti} - c_{\gamma i}) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)}\,, \end{aligned} \tag{7}$$

we now define the gradients $\nabla_{\boldsymbol{z}} x_i^{(n)}$ and Hessians $\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)}$ of Eq. (1) in a recursive way:

$$\nabla_{\boldsymbol{z}} x_i^{(n)} = \boldsymbol{w}_{[n]:i} \cdot x_i^{(n-1)} + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)} \tag{8}$$

$$\begin{aligned} \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)} = \nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top + \{\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top\}^\top \\ + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}\,, \end{aligned} \tag{9}$$

with $\nabla_{\boldsymbol{z}} x_i^{(1)} = \boldsymbol{w}_{[1]:i}$ and $\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(1)} = \boldsymbol{0}_{d\times d}$, being $\boldsymbol{0}_{d\times d}$ a $d \times d$ matrix with 0 in every position.

We are ready to compute a lower bound on the minimum eigenvalue of the Hessian matrix in the feasible set. Firstly, for any $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$ and any polynomial degree N, we can express the set of possible Hessians $\mathcal{H} = \{\boldsymbol{H}_g(\boldsymbol{z}) : \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\}$ as an interval matrix. An interval matrix is a matrix $[\boldsymbol{M}] \in \mathbb{R}^{d\times d\times 2}$ where every position $[m]_{ij} = [\mathcal{L}(m_{ij}), \mathcal{U}(m_{ij})]$ is an interval. Therefore, if $\boldsymbol{H}_g(\boldsymbol{z})$ is bounded for $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$, then we can represent $\mathcal{H} = \{\boldsymbol{H}_g(\boldsymbol{z}) : \boldsymbol{H}_g(\boldsymbol{z}) \in [\boldsymbol{M}]\} = \{\boldsymbol{H}_g(\boldsymbol{z}) : \mathcal{L}(m_{ij}) \leq H_g(\boldsymbol{z})_{ij} \leq \mathcal{U}(m_{ij}), \forall i, j \in [d]\}$.

For every set of Hessians, we can define the lower bounding Hessian $\boldsymbol{L_H}$. Described in Adjiman et al. (1998), This matrix satisfies that $\lambda_{\min}(\boldsymbol{L_H}) \leq \lambda_{\min}(\boldsymbol{H}_g(\boldsymbol{z})), \forall \boldsymbol{H}_g(\boldsymbol{z}) \in \mathcal{H}, \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$. Let $\mathcal{L}(\boldsymbol{M})$ and $\mathcal{U}(\boldsymbol{M})$, the lower bounding Hessian is defined as follows:

$$\boldsymbol{L_H} = \frac{\mathcal{L}(\boldsymbol{M}) + \mathcal{U}(\boldsymbol{M})}{2} + \text{diag}\left(\frac{\mathcal{L}(\boldsymbol{M})\boldsymbol{1} - \mathcal{U}(\boldsymbol{A})\boldsymbol{1}}{2}\right), \tag{10}$$

where $\boldsymbol{1}$ is a vector of ones and $\text{diag}(\boldsymbol{v})$ is a diagonal matrix with the vector $\boldsymbol{v}$ in the diagonal.

Then, we can obtain the spectral radius $\rho(\boldsymbol{L_H})$ with a power method. As the spectral radius satisfies $\rho(\boldsymbol{L_H}) \geq |\lambda_i(\boldsymbol{L_H})|, \forall i \in [d]$, the following inequality holds:

$$\begin{aligned} -\rho(\boldsymbol{L_H}) \leq \lambda_{\min}(\boldsymbol{L_H}) \leq \lambda_{\min}(\boldsymbol{H}_g(\boldsymbol{z})), \\ \forall \boldsymbol{H}_g(\boldsymbol{z}) \in \mathcal{H},\ \ \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\,, \end{aligned} \tag{11}$$

allowing us to use:
$\alpha = \frac{\rho(\boldsymbol{L_H})}{2} \geq \max\{0, -\frac{1}{2}\min\{\lambda_{\min}(\boldsymbol{H}_f(\boldsymbol{z})) : \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\}\}$.

3.3. Efficient power method for spectral radius computation of the lower bounding Hessian

By using interval propagation, one can easily compute sound lower and upper bounds on each position of the Hessian matrix, compute the lower bounding Hessian and perform a power method with it to obtain the spectral radius ρ. However, this method would not scale well to high dimensional scenarios. For instance, in the STL10 case, with 96×96 RGB images ($d = 96 \cdot 96 \cdot 3 = 27,648$) our Hessian matrix would require in the order of $O(d^2) = O(10^9)$ real numbers to be stored. This makes it intractable to perform a power method over such an humongous matrix, or even to compute the lower bounding Hessian. Alternatively, we take advantage of the low rank decomposition characterizing PNs to efficiently perform a power method over the lower bounding Hessian.

Standard power method for spectral radius computation Given any squared and real valued matrix $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ and an initial vector $\boldsymbol{v}_0 \in \mathbb{R}^d$ that is not an eigenvector of $\boldsymbol{M}$, the sequence: $\boldsymbol{v}_n = \frac{\boldsymbol{M}(\boldsymbol{M}\boldsymbol{v}_{n-1})}{||\boldsymbol{M}(\boldsymbol{M}\boldsymbol{v}_{n-1})||_2}$, converges to the eigenvector with the largest eigenvalue in absolute value, i.e. the eigenvector where the spectral norm is attained, being the spectral norm $\rho(\boldsymbol{M}) = ||\boldsymbol{M}(\boldsymbol{M}\boldsymbol{v}_{n-1})||_2$ (Mises & Pollaczek-Geiringer, 1929).

Power method over lower bounding Hessian of PNs

We can employ IBP (Section 3.1) in order to obtain an ex-

pression of the lower bounding Hessian ($\boldsymbol{L_H}$) and evaluate each step of the power method as:

$$\boldsymbol{L_H v} = \frac{\mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} + \mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v}}{2} + \left(\frac{\mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\mathbf{1} - \mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\mathbf{1}}{2}\right) * \boldsymbol{v}\,. \tag{12}$$

Applying IBP on Eq. (7) we obtain:

$$\begin{aligned}\mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} &= \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{+}\,\mathcal{L}(\nabla^2_{\boldsymbol{zz}} x_i^{(N)})\boldsymbol{v} \\ &\quad + \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{-}\,\mathcal{U}(\nabla^2_{\boldsymbol{zz}} x_i^{(N)})\boldsymbol{v} \\ \mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} &= \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{-}\,\mathcal{L}(\nabla^2_{\boldsymbol{zz}} x_i^{(N)})\boldsymbol{v} \\ &\quad + \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{+}\,\mathcal{U}(\nabla^2_{\boldsymbol{zz}} x_i^{(N)})\boldsymbol{v}\,.\end{aligned} \tag{13}$$

We can recursively evaluate $\mathcal{L}(\nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v}$ and $\mathcal{U}(\nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v}$ efficiently as these matrices can be expressed as a sum of rank-1 matrices, check Proposition 1 in the appendix.

Lastly, by applying recursively Proposition 1 from $n = N$ to $n = 1$, starting with $\delta = 1$, we can substitute the results on Eq. (13) and then on Eq. (12) to efficiently evaluate a step of the power method without needing to store the lower bounding Hessian matrix or needing to perform expensive matrix-vector products.

Overall, our lower bounding method consists in computing a valid value of α that satisfies that the α-convexified objective g_α is convex, following Eqs. (4) and (5). In particular, we use $\alpha = \frac{\rho(\boldsymbol{L_H})}{2}$. $\rho(\boldsymbol{L_H})$ is computed via a power method, where the main operation $\boldsymbol{L_H v}$ is evaluated without the need to compute or store the $\boldsymbol{L_H}$ matrix. Provided this valid α, we perform PGD over g_α and this provides a lower bound of the global minima of Eq. (4).

4. Experiments

In this section we show the efficiency of our method by comparing against a simple Black-box solver. Tightness of bounds is also analyzed in comparison with IBP. Finally, a study of the performance of our method in different scenarios is performed. Unless otherwise specified, every network is trained for 100 epochs with Stochastic Gradiend Descent (SGD), with a learning rate of 0.001, which is divided by 10 at epochs $[40, 60, 80]$, momentum 0.9, weight decay $5{\cdot}10^{-5}$ and batch size 128. We thoroughly evaluate our method over the popular image classification datasets MNIST (LeCun et al., 1998), CIFAR10 (Krizhevsky et al., 2014) and STL10 (Coates et al., 2011). Every experiment is done over the first 1000 images of the test dataset, this is a common practice in verification (Singh et al., 2019). For images that are correctly classified by the network, we sequentially verify robustness against the remaining classes in decreasing order of network output. Each verification problem is given a maximum execution time of 60 seconds, we include experiments with different time limits in Appendix E. Note that the execution time can be longer as execution is cut in an asynchronous way, i.e., after we finish the iteration of the BaB algorithm where the time limit is reached. **All of our experiments were conducted on a single-GPU machine**.

4.1. Comparison with Black-box solver

In this experiment, we compare the performance of our BaB verification algorithm with the Black-box solver Gurobi (Gurobi Optimization, LLC, 2022). Gurobi can globally solve Quadratically Constrained Quadratic Programs whether they are convex or not. As this solver cannot extend to higher degree polynomial functions, we train 2^{nd} degree PNs with hidden size $k = 16$ to compare the verification time of our method with Gurobi. In order to do so, we express the verification objective as a quadratic form $g(\boldsymbol{z}) = f(\boldsymbol{z})_t - f(\boldsymbol{z})_a = \boldsymbol{z}^\top \boldsymbol{Q}\boldsymbol{z} + \boldsymbol{q}^\top \boldsymbol{z} + c$ this together with the input constraints $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$ is fed to Gurobi and optimized until convergence.

The black-box solver approach neither scales to higher dimensional inputs nor to higher polynomial degrees. With this approach we need $\mathcal{O}(d^2)$ memory to store the quadratic form, which makes it unfeasible for datasets with higher resolution images than CIFAR10. On the contrary, as seen in Table 1, our approach does not need so much memory and can scale to datasets with larger input sizes like STL10.

4.2. Comparison with IBP

In this experiment we compare the tightness of the lower bounds provided by IBP and α-convexification and their effectiveness when employed for verification. This is done by executing one upper bounding step with PGD and one lower bounding step for IBP and α-convexification methods over the initial feasible set provided by ϵ (see Eq. (3)). We compare the average of the distance from each lower bound to the PGD upper bound over the first 1000 images of the MNIST dataset for PNs with hidden size $k = 25$ and degrees ranging from 2 to 7. We also evaluate verified accuracy of 4^{th} degree PNs (PN_Conv4) with both bounding methods and a maximum time of 120 seconds, for details on the architecture of these networks, we refer to Appendix E. When using IBP, we get a much looser lower bound than with α-convexification, see Fig. 2. Only for high-degree, high-ϵ combinations IBP lower bounds are closer to the

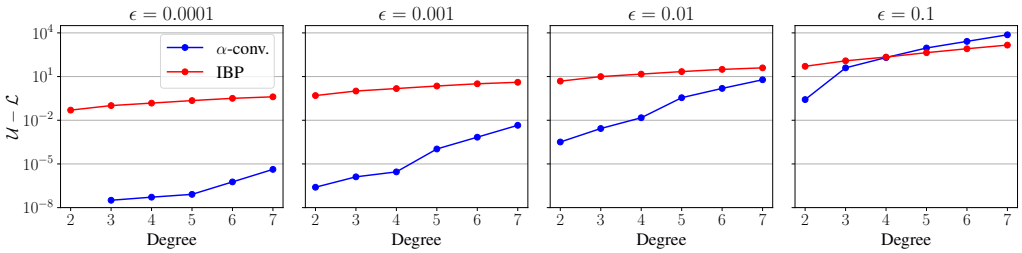

Figure 2: Average difference in log-scale between PGD upper bound ($\mathcal{U}$) and lower bound ($\mathcal{L}$) provided by α-convexification (blue) and IBP (red) of the first 1000 images of the MNIST dataset. α-convexification bounds are significantly tighter than IBP for small ϵ values and all PN degrees from 2 to 7.

Table 1: Verification results for 2×16 PNs. Columns `F`, `T` and `t.o.` refer to the number of images where robustness is falsified, verified and timed-out respectively. When comparing with a black-box solver, our method is much faster and can scale to higher dimensional inputs. This is due to our efficient exploitation of the low-rank factorization of PNs.

Dataset	Correct	ϵ	VPN (Our method)				Gurobi			
			time	F	T	t.o.	time	F	T	t.o.
MNIST	961	0.00725	**1.76**	37	924	0	16.6	37	924	0
$(1 \times 28 \times 28)$		0.013	**1.78**	71	890	0	15.13	71	890	0
		0.05	**1.43**	682	267	12	6.25	691	270	0
		0.06	**1.5**	790	155	16	4.47	799	162	0
CIFAR10	460	1/610	**1.03**	90	370	0	328.0	90	370	0
$(3 \times 32 \times 32)$		1/255	**1.0**	183	277	0	250.07	183	277	0
		4/255	**0.92**	427	28	5	87.93	429	31	0
STL10	362	1/610	**5.06**	142	220	0	out of memory			
$(3 \times 96 \times 96)$		1/255	**3.61**	246	113	3				
		4/255	**1.39**	360	1	1				

Table 2: Verification results for PN_Conv4 PNs, see Table 3. We compare our method employing IBP and α-convexification for lower bounding the objective. Columns `F`, `T` and `t.o.` refer to the number of images where robustness is falsified, verified and timed-out respectively. When using α-convexification bounds we get a low number of timed-out instances, while when using IBP the number of verified instances is 0 for every network-ϵ pair, which makes it unsuitable for PN verification.

Dataset	ϵ	IBP				VPN (α-convexification)			
		Time(s)	F	T	t.o.	Time(s)	F	T	t.o.
MNIST	0.015	0.3	22	0	964	50	22	**963**	1
	0.026	0.4	38	0	948	69	38	**929**	19
CIFAR10	1/255	0.4	158	0	468	274.6	159	**455**	12
	2/255	0.5	321	0	305	224.1	321	**165**	140
STL10*	1/255	3.4	79	0	114	2481.0	79	**112**	2

*Note: Results obtained in the first 500 images of the dataset due to the longer running times because of the larger input size of STL10.

PGD upper bound. In practice, this is not a problem for verification, as for epsilons in the order of 0.1, it is really easy to find adversarial examples with PGD and there will be no accuracy left to verify.

The looseness of the IBP lower bound is confirmed when comparing the number of verified images with IBP and α-convexification, see Table 2. With the latter, we are able to verify the accuracy of 4^{th} degree PNs almost exactly (almost no timed-out instances) in every studied dataset, while with the former, we are not able to verify robustness for a single image in any network-ϵ pair, confirming the fact that IBP cannot be used for PN verification.

5. Conclusion

We propose a method to verify polynomial networks (PNs). Our method, which can be categorized as a α-BaB global optimization algorithm, is a novel approach to verification of PNs. We believe can be extended to cover other twice-differentiable networks in the future. We exhibit that our method outperforms existing methods, such as black-box solvers and IBP. Our method enables verification in datasets

such as STL10, which includes RGB images of 96×96 resolution. This is larger than the images typically used in previous verification methods. Our method can further encourage the community to extend verification to a broader class of functions as well as conduct experiments in datasets of higher resolution. Our IBP bounding method is related to NN IBP methods and could be possibly extended to symbolic interval propagation (Wang et al., 2018a; Henriksen & Lomuscio, 2020; 2021).

6. Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 725594 - time-data). This work was supported by the Swiss National Science Foundation (SNSF) under grant number 200021_205011 (**SNF project – Deep Optimisation**). This work was supported by the Swiss National Science Foundation (SNSF) under grant number 200021_178865 (**SNF project - Theory and Methods for Storage-Optimal Optimization**). This work was supported by Zeiss.

References

Adjiman, C. S. and Floudas, C. A. Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9(1):23–40, Jul 1996. ISSN 1573-2916. doi: 10.1007/BF00121749. URL https://doi.org/10.1007/BF00121749.

Adjiman, C. S., Dallwig, S., Floudas, C. A., and Neumaier, A. A global optimization method, αbb, for general twice-differentiable constrained nlps — i. theoretical advances. *Computers & Chemical Engineering*, 22:1137–1158, 1998.

Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. *CoRR*, abs/1904.09959, 2019. URL http://arxiv.org/abs/1904.09959.

Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A. V., and Criminisi, A. Measuring neural net robustness with constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2621–2629, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Bunel, R., Lu, J., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020a. URL http://jmlr.org/papers/v21/19-468.html.

Bunel, R., Palma, A. D., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P. H. S., and Kumar, M. P. Lagrangian decomposition for neural network verification. *CoRR*, abs/2002.10410, 2020b. URL https://arxiv.org/abs/2002.10410.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017. doi: 10.1109/SP.2017.49.

Choraria, M., Dadi, L. T., Chrysos, G., Mairal, J., and Cevher, V. The spectral bias of polynomial neural networks. In *International Conference on Learning Representations (ICLR)*, 2022.

Chrysos, G., Moschoglou, S., Bouritsas, G., Panagakis, Y., Deng, J., and Zafeiriou, S. $\pi-$nets: Deep polynomial neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Chrysos, G. G. and Panagakis, Y. Naps: Non-adversarial polynomial synthesis. *Pattern Recognition Letters*, 140:318–324, 2020. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2020.11.006. URL https://www.sciencedirect.com/science/article/pii/S0167865520304116.

Chrysos, G. G., Georgopoulos, M., Deng, J., and Panagakis, Y. Polynomial networks in deep classifiers. *CoRR*, abs/2104.07916, 2021a. URL https://arxiv.org/abs/2104.07916.

Chrysos, G. G., Moschoglou, S., Bouritsas, G., Deng, J., Panagakis, Y., and Zafeiriou, S. P. Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021b. ISSN 1939-3539. doi: 10.1109/tpami.2021.3058891. URL http://dx.doi.org/10.1109/TPAMI.2021.3058891.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In Gordon, G., Dunson, D., and Dudík, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/coates11a.html.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Dong, Y., Su, H., Wu, B., Li, Z., Liu, W., Zhang, T., and Zhu, J. Efficient decision-based black-box adversarial attacks on face recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7714–7722, 2019.

Dou, Z., Osher, S. J., and Wang, B. Mathematical analysis of adversarial attacks. *arXiv preprint arXiv:1811.06492*, 2018.

Ehlers, R. Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320, 2017. URL http://arxiv.org/abs/1705.01320.

Fan, F., Li, M., Wang, F., Lai, R., and Wang, G. Expressivity and trainability of quadratic networks. *CoRR*, abs/2110.06081, 2021. URL https://arxiv.org/abs/2110.06081.

Ferrari, C., Mueller, M. N., Jovanović, N., and Vechev, M. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=l_amHf1oaK.

Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and 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, 2018. doi: 10.1109/SP.2018.00058.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

Goswami, G., Agarwal, A., Ratha, N. K., Singh, R., and Vatsa, M. Detecting and mitigating adversarial perturbations for robust face recognition. *International Journal of Computer Vision (IJCV)*, 127:719–742, 2019.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

Henriksen, P. and Lomuscio, A. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI*, 2020.

Henriksen, P. and Lomuscio, A. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In Zhou, Z.-H. (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2549–2555. International Joint Conferences on Artificial Intelligence Organization, 8 2021. URL https://doi.org/10.24963/ijcai.2021/351. Main Track.

Horst, R. and Tuy, H. *Global Optimization: Deterministic Approaches*. Springer, Berlin, Heidelberg, 1996. URL https://doi.org/10.1007/978-3-662-03199-5_4.

Katz, G., Barrett, C., Dill, D., Julian, K., and Kochenderfer, M. Reluplex: An efficient smt solver for verifying deep neural networks, 2017. URL https://arxiv.org/abs/1702.01135.

Kelley, C. *5. Simple Bound Constraints*, pp. 87–108. 1999. doi: 10.1137/1.9781611970920.ch5. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611970920.ch5.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Krizhevsky, A., Nair, V., and Hinton, G. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

Land, A. H. and Doig, A. G. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1910129.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Li, Q., Guo, Y., and Chen, H. Practical no-box adversarial attacks against dnns. In *Advances in neural information processing systems (NeurIPS)*, 2020.

Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., and Kochenderfer, M. J. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021. ISSN 2167-3888. doi: 10.1561/2400000035. URL http://dx.doi.org/10.1561/2400000035.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Mises, R. V. and Pollaczek-Geiringer, H. Praktische verfahren der gleichungsauflösung . *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für*

Angewandte Mathematik und Mechanik, 9(2):152–164, 1929. doi: https://doi.org/10.1002/zamm.19290090206. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.19290090206.

Moore, R. E., Kearfott, R. B., and Cloud, M. J. Introduction to interval analysis. 2009.

Palma, A. D., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P. H. S., and Kumar, M. P. Improved branch and bound for neural network verification via lagrangian decomposition. *CoRR*, abs/2104.06718, 2021. URL https://arxiv.org/abs/2104.06718.

Raghunathan, A., Xie, S. M., Yang, F., Duchi, J., and Liang, P. Understanding and mitigating the tradeoff between robustness and accuracy. In *International Conference on Machine Learning*, pp. 7909–7919. PMLR, 2020.

Royo, V. R., Calandra, R., Stipanović, D. M., and Tomlin, C. J. Fast neural network verification via shadow prices. *ArXiv*, abs/1902.07247, 2019.

Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lWUoA9FQ.

Singh, G., Ganvir, R., Püschel, M., and Vechev, M. Beyond the single neuron convex barrier for neural network certification. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/0a9fdbb17feb6ccb7ec405cfb85222c4-Paper.pdf.

Stone, M. H. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(4):167–184, 1948. ISSN 0025570X, 19300980. URL http://www.jstor.org/stable/3029750.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199.

Tjeng, V., Xiao, K. Y., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGIdiRqtm.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pp. 1599–1614, USA, 2018a. USENIX Association. ISBN 9781931971461.

Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., and Kolter, J. Z. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *CoRR*, abs/2103.06624, 2021. URL https://arxiv.org/abs/2103.06624.

Wang, X., Girshick, R., Gupta, A., and He, K. Non-local neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, 2018b. doi: 10.1109/CVPR.2018.00813.

Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., and Jordan, M. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pp. 7472–7482. PMLR, 2019.

Zhong, Y. and Deng, W. Adversarial learning with margin-based triplet embedding regularization. In *International Conference on Computer Vision (ICCV)*, pp. 6549–6558, 2019.

Zhu, Z., Latorre, F., Chrysos, G., and Cevher, V. Controlling the complexity and lipschitz constant improves polynomial nets. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=dQ7Cy_ndl1s.

Contents of the appendix

We include additional experiments and ablation studies in Appendix E. Appendix B provides a more detailed coverage of PN architectures. In Appendix C, we include the pseudocode of our algorithms and we provide an analysis of the complexity. To conclude, in Appendix D, we include all of our proofs.

A. Related Work

In this section, we give an overview of neural network verification and polynomial networks, that are centered around our target in this work.

A.1. Neural Network Verification

Early works on sound and complete NN verification were based on Mixed Integer Linear Programming (MILP) and Satisfiability Modulo Theory (SMT) solvers (Katz et al., 2017; Ehlers, 2017; Bastani et al., 2016; Tjeng et al., 2019) and were limited to both small datasets and networks.

The utilization of custom BaB algorithms enabled verification to scale to datasets and networks that are closer to those used in practice. Bunel et al. (2020a) review earlier methods like Katz et al. (2017) and show they can be formulated as BaB algorithms. BaDNB (Palma et al., 2021) proposes a novel branching strategy called Filtered Smart Branching and uses the Lagrangian decomposition-based bounding algorithm proposed in Bunel et al. (2020b). β-CROWN (Wang et al., 2021) proposes a bound propagation based algorithm. MN-BaB (Ferrari et al., 2022) proposes a cost adjusted branching strategy and leverages multi-neuron relaxations and a GPU-based solver for bounds computing. Our work centers on the bounding algorithm by proposing a general convex lowerbound adapted to PNs.

BaB algorithms for ReLU networks focus their branching strategies on the activity of ReLU neurons. This has been observed to work better than input set branching for ReLU networks (Bunel et al., 2020a). Similarly to our method, Anderson et al. (2019), Wang et al. (2018a), Royo et al. (2019) use input set branching strategies.

A.2. Polynomial Networks

First works have been focused on developing the foundations and showcasing the performance of PNs in different tasks (Chrysos et al., 2021b; Chrysos & Panagakis, 2020). Also, in Chrysos et al. (2021a), PN classifiers are formulated in a common framework where other previous methods like Wang et al. (2018b) can be framed. Lately, more emphasis has been put onto proving theoretical properties of PNs (Fan et al., 2021; Choraria et al., 2022). In Zhu et al. (2022), they derive Lipschitz constant and complexity bounds for two PN decompositions in terms of the l_∞ and l_2 norms. They also analyze robustness of PNs against PGD adversarial attacks by measuring percentage of images where PGD fails to find an adversarial example, which is a complete but not sound verification method. Our verification method is sound and complete.

B. Background

B.1. Coupled CP decomposition (CCP)

Relying on the CP decomposition (Kolda & Bader, 2009), the CCP decomposition provides us a core expression of PNs as used in Chrysos & Panagakis (2020) to construct a generative model. Let N be the polynomial degree, $\boldsymbol{z} \in \mathbb{R}^d$ the input vector, d, k and o the input, hidden and output sizes, the CCP decomposition can be expressed as:

$$\boldsymbol{x}^{(n)} = (\boldsymbol{W}_{[n]}^\top \boldsymbol{z}) * \boldsymbol{x}^{(n-1)} + \boldsymbol{x}^{(n-1)}, \forall n = 2, 3, \dots, N, \tag{14}$$

where $\boldsymbol{x}^{(1)} = \boldsymbol{W}_{[1]}^\top \boldsymbol{z}$, $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{x}^{(N)} + \boldsymbol{\beta}$ and $*$ denotes the Hadamard product.

For example, the second order CCP factorization will lead to the following formulation:

$$\boldsymbol{x}^{(1)} = \boldsymbol{W}_{[1]}^\top \boldsymbol{z}, \quad \boldsymbol{x}^{(2)} = \boldsymbol{W}_{[2]}^\top \boldsymbol{z} * \boldsymbol{x}^{(1)} + \boldsymbol{x}^{(1)}, \quad \boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{x}^{(2)} + \boldsymbol{\beta}, \tag{15}$$

where $\boldsymbol{W}_{[1]} \in \mathbb{R}^{d \times k}$, $\boldsymbol{W}_{[2]} \in \mathbb{R}^{d \times k}$ and $\boldsymbol{C} \in \mathbb{R}^{o \times k}$ are weight matrices, $\boldsymbol{\beta} \in \mathbb{R}^o$ is a vector.

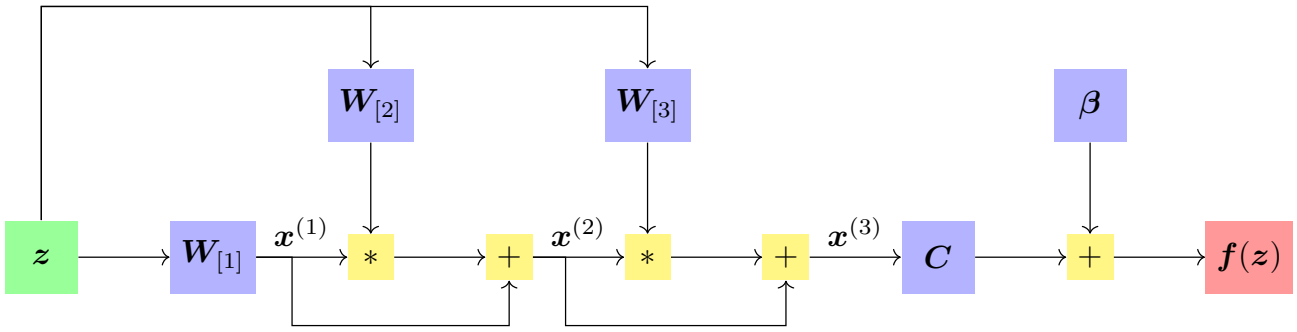

Figure 3: Third degree PN architecture. Blue boxes depict learnable parameters, yellow depict mathematical operations, the green and red boxes are the input and the output respectively. Note that no activation functions are involved, only element-wise (Hadamard) products $*$ and additions $+$. This figure represents the recursive formula of Eq. (1).

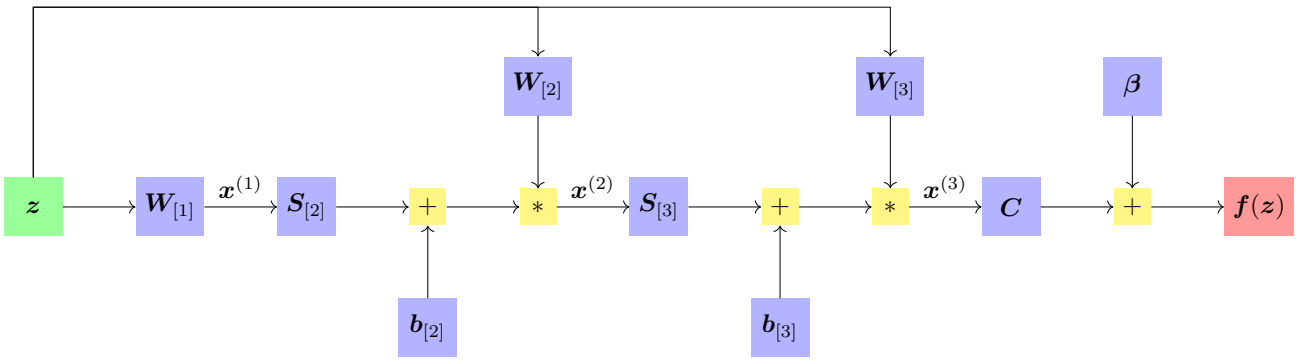

Figure 4: Third order NCP network architecture.

B.2. Nested coupled CP decomposition (NCP)

The NCP model leverages a joint hierarchical decomposition, which provided strong results in both generative and discriminative tasks in Chrysos et al. (2020; 2021b). It can be expressed with the following recursive relation:

$$\boldsymbol{x}^{(n)} = (\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) \,, \tag{16}$$

for $n \in [N-1]+1$ with $\boldsymbol{x}^{(1)} = \boldsymbol{W}_{[1]}^{\top}\boldsymbol{z}$ and $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{x}^{(N)} + \boldsymbol{\beta}$.

We present the first and second order partial derivatives of NCP below.

$$\nabla_{\boldsymbol{z}} x_i^{(n)} = \boldsymbol{w}_{[n]:i} \cdot (\boldsymbol{s}_{[n]:i}^{\top}\boldsymbol{x}^{(n-1)} + b_{[n]i}) + (\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}) \cdot (\sum_{j=1}^{k} s_{[n]ji} \nabla_{\boldsymbol{z}} x_j^{(n-1)}) \tag{17}$$

$$\begin{aligned} \nabla_{\boldsymbol{z}\boldsymbol{z}}^2 x_i^{(n)} = \boldsymbol{w}_{[n]:i} (\sum_{j=1}^{k} s_{[n]ji} \nabla_{\boldsymbol{z}} x_j^{(n-1)})^{\top} + (\sum_{j=1}^{k} s_{[n]ji} \nabla_{\boldsymbol{z}} x_j^{(n-1)}) \boldsymbol{w}_{[n]:i}^{\top} \\ + (\boldsymbol{w}_{[n]:i}^{\top} z) \cdot (\sum_{j=1}^{k} s_{[n]ji} \nabla_{\boldsymbol{z}\boldsymbol{z}}^2 x_j^{(n-1)}) \,. \end{aligned} \tag{18}$$

Our method can be easily extended to this type of PNs. As a reference, we obtain a verified accuracy of 76.2% with an upper bound of 76.4% with an 2×25 NCP at $\epsilon = 0.026$.

B.3. Product of Polynomials

In practice, Chrysos et al. (2021b) report that to reduce further the parameters they are often stacking sequentially a number of polynomials, see Fig. 5. That results in a setting that is referred to as product of polynomials, with the highest degree of

expansion being defined as the product of all the degrees of the individual polynomials. Hopefully, as we will demonstrate below, our formulation can be extended to this setting.

Let $\boldsymbol{x} = \boldsymbol{x}^{(N_1)}$ be the output of a PN $f_1 : \mathbb{R}^d \to \mathbb{R}^k$ of N_1^{th}-degree and $\boldsymbol{y} = \boldsymbol{y}^{(N_2)}$ be the output of a PN $f_2 : \mathbb{R}^k \to \mathbb{R}^k$ of N_2^{th}-degree, with $\boldsymbol{x}$ and $\boldsymbol{y}$ coming either from Eq. (14) or Eq. (16). The product of polynomials f_i and f_2 is defined as $f : \mathbb{R}^d \to \mathbb{R}^o$:

$$\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{f}_2(\boldsymbol{f}_1(\boldsymbol{z})) + \boldsymbol{\beta}\,, \tag{19}$$

where $\boldsymbol{C} \in \mathbb{R}^{o\times k}$ and $\boldsymbol{\beta} \in \mathbb{R}^o$.

We present the first and second order partial derivatives of the Product of polynomials below.

Let $\boldsymbol{z}$ be the input, $\boldsymbol{x} = \boldsymbol{f}_1(\boldsymbol{z})$ and $\boldsymbol{y} = \boldsymbol{f}_2(\boldsymbol{x})$ by applying the chain rule of partial derivatives, we can obtain:

$$\nabla_{\boldsymbol{z}} y_i = \sum_{j=i}^{k} \frac{\partial y_i}{\partial x_j} \nabla_{\boldsymbol{z}} x_j, \quad \nabla^2_{\boldsymbol{z}\boldsymbol{z}} y_i = \sum_{j=i}^{k} \frac{\partial y_i}{\partial x_j} \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_j + \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}) \nabla^2_{\boldsymbol{x}\boldsymbol{x}} y_i \boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\,, \tag{20}$$

where $\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}) \in \mathbb{R}^{k\times d}$ is the Jacobian matrix.

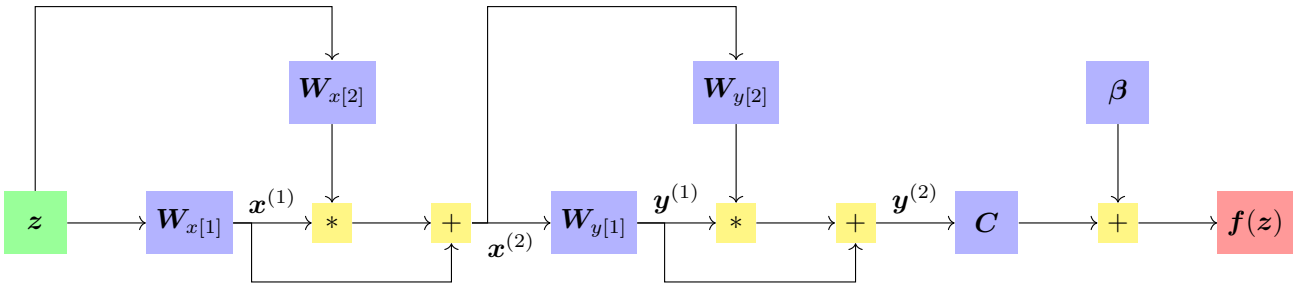

Figure 5: Product of two 2nd-degree CCP polynomials. The final classification layer of the first polynomial is dropped, the output $\boldsymbol{x}^{(2)}$ is fed into a second 2nd-degree polynomial.

B.4. Convolutional PNs (C-PNs)

As seen in Chrysos et al. (2020), the performance of PNs is boosted when employing convolution operators instead of standard linear mappings. Convolutions reduce the number of parameters and take advantage of the local 2D structure of the images. Let $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{c\times h\times w}$ be the input image with c, h and w being the number of channels, height and width respectively. The N^{th}-degree **CCP_Conv** becomes:

$$\boldsymbol{X}^{(n,i)} = (\boldsymbol{\mathcal{Z}} \circ \boldsymbol{\mathcal{W}}_{[n,i]}) * \boldsymbol{X}^{(n-1,i)} + \boldsymbol{X}^{(n-1,i)}\,, \forall n \in [N-1]+1, \forall i \in [q]\,, \tag{21}$$

where $\boldsymbol{X}^{(1,i)} = \boldsymbol{\mathcal{Z}} \circ \boldsymbol{\mathcal{W}}_{[1,i]}, \forall i \in [q]$, $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{C}\boldsymbol{x}^{(N)} + \boldsymbol{\beta}$ and $\boldsymbol{x}^{(N)} = \text{flat}(\boldsymbol{\mathcal{X}}^{(N)})$.

In order to verify C-PNs, we convert every convolutional layer into a linear layer via Toeplitz matrices (see Gehr et al. (2018)) resulting in an equivalent CCP PN.

B.5. Details on IBP

In this section we further elaborate on the IBP for the network outputs, the gradients and Hessians by applying the notions in Section 3.1.

IBP for network outputs In order to compute lower and upper bounds on the output of the network $\boldsymbol{f}(\boldsymbol{z})$, we apply interval arithmetic techniques in the recursive formulas Eqs. (1) and (16). For both the CCP and the NCP cases, let $\hat{\boldsymbol{x}}^{(n)} = \boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}$:

$$\begin{aligned} \mathcal{L}(\hat{\boldsymbol{x}}^{(n)}) &= {\boldsymbol{W}_{[n]}^{\top}}^{+}\boldsymbol{l} + {\boldsymbol{W}_{[n]}^{\top}}^{-}\boldsymbol{u} \\ \mathcal{U}(\hat{\boldsymbol{x}}^{(n)}) &= {\boldsymbol{W}_{[n]}^{\top}}^{+}\boldsymbol{u} + {\boldsymbol{W}_{[n]}^{\top}}^{-}\boldsymbol{l}\,. \end{aligned} \tag{22}$$

In the case of CCP PNs, the recursive formula in Eq. (1) becomes $\boldsymbol{x}^{(n)} = (\hat{\boldsymbol{x}}^{(n)} + \mathbf{1}) * \boldsymbol{x}^{(n-1)}$, whereas for NCP, Eq. (16) becomes $\boldsymbol{x}^{(n)} = (\hat{\boldsymbol{x}}^{(n)}) * (\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]})$ for any $n \in [N-1]+1$, for $n = 1$, in both architectures $\boldsymbol{x}^{(n)} = \hat{\boldsymbol{x}}^{(n)}$. Then, we can define the bounds with the following recursive formulas:

$$
\begin{aligned}
\textbf{CCP} \begin{cases}
S = \begin{Bmatrix}
(\mathcal{L}(\hat{\boldsymbol{x}}^{(n)}) + 1) * \mathcal{L}(\boldsymbol{x}^{(n-1)}), \\
(\mathcal{L}(\hat{\boldsymbol{x}}^{(n)}) + 1) * \mathcal{U}(\boldsymbol{x}^{(n-1)}), \\
(\mathcal{U}(\hat{\boldsymbol{x}}^{(n)}) + 1) * \mathcal{L}(\boldsymbol{x}^{(n-1)}), \\
(\mathcal{U}(\hat{\boldsymbol{x}}^{(n)}) + 1) * \mathcal{U}(\boldsymbol{x}^{(n-1)}),
\end{Bmatrix} \\
\mathcal{L}(\boldsymbol{x}^{(n)}) = \min(S) \\
\mathcal{U}(\boldsymbol{x}^{(n)}) = \max(S)
\end{cases} \\
\textbf{NCP} \begin{cases}
\mathcal{L}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) = {\boldsymbol{S}_{[n]}^{\top}}^{+}\mathcal{L}(\boldsymbol{x}^{(n-1)}) + {\boldsymbol{S}_{[n]}^{\top}}^{-}\mathcal{U}(\boldsymbol{x}^{(n-1)}) + \boldsymbol{b}_{[n]} \\
\mathcal{U}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) = {\boldsymbol{S}_{[n]}^{\top}}^{+}\mathcal{U}(\boldsymbol{x}^{(n-1)}) + {\boldsymbol{S}_{[n]}^{\top}}^{-}\mathcal{L}(\boldsymbol{x}^{(n-1)}) + \boldsymbol{b}_{[n]} \\
S = \begin{Bmatrix}
\mathcal{L}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) * \mathcal{L}(\boldsymbol{x}^{(n-1)}), \\
\mathcal{L}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) * \mathcal{U}(\boldsymbol{x}^{(n-1)}), \\
\mathcal{U}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) * \mathcal{L}(\boldsymbol{x}^{(n-1)}), \\
\mathcal{U}(\boldsymbol{S}_{[n]}^{\top}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) * \mathcal{U}(\boldsymbol{x}^{(n-1)}),
\end{Bmatrix} \\
\mathcal{L}(\boldsymbol{x}^{(n)}) = \min(S) \\
\mathcal{U}(\boldsymbol{x}^{(n)}) = \max(S)\,,
\end{cases}
\end{aligned}
\tag{23}
$$

where the $\min$ and $\max$ operators are applied element-wise in sets of vectors or matrices. Finally, the output bounds are obtained with:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{f}(\boldsymbol{z})) &= \boldsymbol{C}^{+}\,\mathcal{L}(\boldsymbol{x}^{(N)}) + \boldsymbol{C}^{-}\,\mathcal{U}(\boldsymbol{x}^{(N)}) + \boldsymbol{\beta} \\
\mathcal{U}(\boldsymbol{f}(\boldsymbol{z})) &= \boldsymbol{C}^{+}\,\mathcal{U}(\boldsymbol{x}^{(N)}) + \boldsymbol{C}^{-}\,\mathcal{L}(\boldsymbol{x}^{(N)}) + \boldsymbol{\beta}\,.
\end{aligned}
\tag{24}
$$

These operations can be implemented as a forward pass through the PN.

IBP for gradients

The next step is to obtain bounds on the gradients of the PNs. Again, with the help of interval arithmetic theory, we can extend the recursive formulas Eqs. (8) and (17) for computing IBP bounds. In the case of CCP PNs:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n)})) &= \mathcal{L}(\boldsymbol{W}_{[n]} * \boldsymbol{x}^{(n-1)} + (\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
&= \mathcal{L}(\boldsymbol{W}_{[n]} * \boldsymbol{x}^{(n-1)}) + \mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
&= \boldsymbol{W}_{[n]}^{+} * \mathcal{L}(\boldsymbol{x}^{(n-1)}) + \boldsymbol{W}_{[n]}^{-} * \mathcal{U}(\boldsymbol{x}^{(n-1)}) + \mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n)})) &= \mathcal{U}(\boldsymbol{W}_{[n]} * \boldsymbol{x}^{(n-1)} + (\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))) \\
&= \mathcal{U}(\boldsymbol{W}_{[n]} * \boldsymbol{x}^{(n-1)}) + \mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
&= \boldsymbol{W}_{[n]}^{+} * \mathcal{U}(\boldsymbol{x}^{(n-1)}) + \boldsymbol{W}_{[n]}^{-} * \mathcal{L}(\boldsymbol{x}^{(n-1)}) + \mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\,,
\end{aligned}
\tag{25}
$$

with

$$
\begin{aligned}
S &= \begin{Bmatrix}
\mathcal{L}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})), \\
\mathcal{L}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})), \\
\mathcal{U}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})), \\
\mathcal{U}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))
\end{Bmatrix} \\
\mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \min(S) \\
\mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z} + 1) * \boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \max(S)\,,
\end{aligned}
\tag{26}
$$

where the Hadamard product of a $\mathbb{R}^{k\times d}$ matrix with a $\mathbb{R}^d$ vector results in a $\mathbb{R}^{k\times d}$ matrix. The $\min$ and $\max$ operators are applied element-wise in sets of vectors or matrices.

In the case of NCP PNs:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n)})) &= \mathcal{L}(\boldsymbol{W}_{[n]} * (\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) + (\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})))\\
&= \mathcal{L}(\boldsymbol{W}_{[n]} * (\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]})) + \mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})))\\
&= \boldsymbol{W}_{[n]}^{+} * \mathcal{L}(\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]})) + \boldsymbol{W}_{[n]}^{-} * \mathcal{U}(\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}))\\
&\quad + \mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})))\\
\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n)})) &= \mathcal{U}(\boldsymbol{W}_{[n]} * (\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}) + (\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})))\\
&= \mathcal{U}(\boldsymbol{W}_{[n]} * (\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]})) + \mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})))\\
&= \boldsymbol{W}_{[n]}^{+} * \mathcal{U}(\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]})) + \boldsymbol{W}_{[n]}^{-} * \mathcal{L}(\boldsymbol{S}_{[n]}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}_{[n]}))\\
&\quad + \mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))),
\end{aligned} \tag{27}$$

with

$$\begin{aligned}
\mathcal{L}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \boldsymbol{S}_{[n]}^{+}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) + \boldsymbol{S}_{[n]}^{-}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\\
\mathcal{U}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \boldsymbol{S}_{[n]}^{+}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) + \boldsymbol{S}_{[n]}^{-}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\\
S &= \begin{Bmatrix}
\mathcal{L}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * \mathcal{L}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})),\\
\mathcal{L}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * \mathcal{U}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})),\\
\mathcal{U}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * \mathcal{L}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})),\\
\mathcal{U}(\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * \mathcal{U}(\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))
\end{Bmatrix}\\
\mathcal{L}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))) &= \min(S)\\
\mathcal{U}((\boldsymbol{W}_{[n]}^{\top}\boldsymbol{z}) * (\boldsymbol{S}_{[n]}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))) &= \max(S)\,.
\end{aligned} \tag{28}$$

Eq. (25) (Eq. (27)) jointly with Eq. (26) (Eq. (28)) allows to recursively obtain gradient bounds for the CCP (NCP) PNs starting with $\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(1)})) = \mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(1)})) = \boldsymbol{W}_{[1]}$. Finally, bounds of the gradients of the network output in both CCP and NCP cases are:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{f})) &= \boldsymbol{C}^{+}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}^{(N)})) + \boldsymbol{C}^{-}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}^{(N)})) + \boldsymbol{\beta}\\
\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{f})) &= \boldsymbol{C}^{+}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}^{(N)})) + \boldsymbol{C}^{-}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}^{(N)})) + \boldsymbol{\beta}\,.
\end{aligned} \tag{29}$$

IBP for Hessians Lastly, with help of interval arithmetic and aforementioned IBP for PN outputs and gradients, we can use recursive formulas Eqs. (9) and (18) to compute bounds of the Hessian matrices.

In the case of CCP PNs:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n)})) &= \mathcal{L}(\nabla_{\boldsymbol{z}}x_i^{(n-1)}\boldsymbol{w}_{[n]:i}^{\top} + \{\nabla_{\boldsymbol{z}}x_i^{(n-1)}\boldsymbol{w}_{[n]:i}^{\top}\}^{\top} + (\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)}))\\
&= \mathcal{L}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{+\top} + \mathcal{U}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{-\top}\\
&\quad + \boldsymbol{w}_{[n]:i}^{+}\mathcal{L}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})^{\top} + \boldsymbol{w}_{[n]:i}^{-}\mathcal{U}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})^{\top}\\
&\quad + \mathcal{L}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)}))\\
\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n)})) &= \mathcal{U}(\nabla_{\boldsymbol{z}}x_i^{(n-1)}\boldsymbol{w}_{[n]:i}^{\top} + \{\nabla_{\boldsymbol{z}}x_i^{(n-1)}\boldsymbol{w}_{[n]:i}^{\top}\}^{\top} + (\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)}))\\
&= \mathcal{U}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{+\top} + \mathcal{L}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{-\top}\\
&\quad + \boldsymbol{w}_{[n]:i}^{+}\mathcal{U}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})^{\top} + \boldsymbol{w}_{[n]:i}^{-}\mathcal{L}(\nabla_{\boldsymbol{z}}x_i^{(n-1)})^{\top}\\
&\quad + \mathcal{U}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)}))\,,
\end{aligned} \tag{30}$$

where

$$
\begin{aligned}
S = \left\{ \begin{array}{l} \mathcal{L}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})), \\ \mathcal{L}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})), \\ \mathcal{U}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})), \\ \mathcal{U}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})) \end{array} \right\} \\
\mathcal{L}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})) = \min(S) \\
\mathcal{U}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z}+1)\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n-1)})) = \max(S)\,.
\end{aligned}
\tag{31}
$$

In the case of NCP PNs:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n)})) &= \mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top} + \{\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}\}^{\top} + (\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) \\
&= \mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\boldsymbol{w}_{[n]:i}^{+\top} + \mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\boldsymbol{w}_{[n]:i}^{l\top} \\
&+ \boldsymbol{w}_{[n]:i}^{+}\mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))^{\top} + \boldsymbol{w}_{[n]:i}^{-}\mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))^{\top} \\
&+ \mathcal{L}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) \\
\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(n)})) &= \mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top} + \{\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}\}^{\top} + (\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) \\
&= \mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\boldsymbol{w}_{[n]:i}^{+\top} + \mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))\boldsymbol{w}_{[n]:i}^{l\top} \\
&+ \boldsymbol{w}_{[n]:i}^{+}\mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))^{\top} + \boldsymbol{w}_{[n]:i}^{-}\mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)}))^{\top} \\
&+ \mathcal{U}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)}))\,,
\end{aligned}
\tag{32}
$$

where

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \boldsymbol{s}_{[n]i}^{+}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) + \boldsymbol{s}_{[n]i}^{-}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
\mathcal{U}(\boldsymbol{s}_{[n]i}\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) &= \boldsymbol{s}_{[n]i}^{+}\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) + \boldsymbol{s}_{[n]i}^{-}\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}^{\top}(\boldsymbol{x}^{(n-1)})) \\
\mathcal{L}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) &= \sum_{j=1}^{k} s_{ij}^{+}\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) + s_{ij}^{-}\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) \\
\mathcal{U}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) &= \sum_{j=1}^{k} s_{ij}^{+}\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) + s_{ij}^{-}\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)}))\,,
\end{aligned}
\tag{33}
$$

and

$$S = \left\{ \begin{array}{l} \mathcal{L}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\,\mathcal{L}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})), \\ \mathcal{L}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\,\mathcal{U}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})), \\ \mathcal{U}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\,\mathcal{L}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})), \\ \mathcal{U}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\,\mathcal{U}(\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) \end{array} \right\} \tag{34}$$

$$\mathcal{L}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) = \min(S)$$

$$\mathcal{U}((\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z})\sum_{j=1}^{k} s_{ij}\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(n-1)})) = \max(S)\,.$$

Similarly to gradient bounds, we can recursively compute $\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(N)}))$ and $\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(N)}))$ starting with $\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(1)})) = \mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_i^{(1)})) = \mathbf{0}_{d\times d}$. Finally, bounds of the Hessian matrix of the output can be obtained with:

$$\begin{aligned} \mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(f(\boldsymbol{z})_i)) &= \sum_{j=1}^{k} c_{ij}^{+}\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(N)})) + c_{ij}^{-}\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(N)})) \\ \mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(f(\boldsymbol{z})_i)) &= \sum_{j=1}^{k} c_{ij}^{+}\,\mathcal{U}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(N)})) + c_{ij}^{-}\,\mathcal{L}(\boldsymbol{H}_{\boldsymbol{z}}(x_j^{(N)}))\,. \end{aligned} \tag{35}$$

C. BaB algorithm for PN robustness verification

BaB algorithms (Land & Doig, 1960) are a well known approach to global optimization (Horst & Tuy, 1996). These algorithms intend to find the global minima of a optimization problem in the form of Eq. (4), therefore guaranteeing soundness and completeness for verification. In this section we present the details of our BaB based verification algorithm, we prove the theoretical convergence of BaB to the global minima of Eq. (4) and the complexity of its key steps.

Firstly, the property that we want to verify or falsify is the one given by Eq. (2). This property is defined by (i) the network f, (ii) the adversarial budget ϵ, (iii) a correctly classified input $\boldsymbol{z}_0 : \arg\max \boldsymbol{f}(\boldsymbol{z}_0) = t$. In order to verify the property, it is necessary that for each $\gamma \neq t$, the global minima of Eq. (4) is greater than 0, i.e., $\forall\gamma \neq t : v^* = \min_{\boldsymbol{z}\in C_{\text{in}}} f(\boldsymbol{z})_t - f(\boldsymbol{z})_\gamma > 0$. On the contrary, in order to falsify Eq. (2), it is sufficient that for any $\gamma \neq t$ the global minima of Eq. (4) is smaller or equal than 0, i.e., $\exists\gamma \neq t : v^* = \min_{\boldsymbol{z}\in C_{\text{in}}} f(\boldsymbol{z})_t - f(\boldsymbol{z})_\gamma \leq 0$. In order to reduce execution times, we heuristically sort all the γ in decreasing order by network output, $\{\gamma_i : \gamma_i \neq t, f(\boldsymbol{z}_0)_{\gamma_i} \geq f(\boldsymbol{z}_0)_{\gamma_j} \forall j > i\}$ and solve Eq. (4) until one global minima is smaller or equal to 0 or all global minimas are greater than 0.

In order to solve Eq. (4), we use Algorithm 1. Without any modifications, this algorithm converges to the global minima. However, for verification, it is sufficient to find that the lower bound of the global minima (`global_lb`) cannot be smaller than or equal to 0, or that the upper bound of the global minima in a subset (`prob_ub`) is smaller than 0, i.e., there exists an adversarial example in that subset. Therefore we can cut the execution early when employed for verification, these optional changes are highlighted in red in Algorithm 1.

Algorithm 1 can theoretically be applied to any twice-differentiable classifier provided we have a method for computing an α for obtaining valid α-convexification bounds. For this matter, we propose Algorithm 2, which again can be used for any twice-differentiable classifier if we are able to compute its lower bounding Hessian $\boldsymbol{L}_{\boldsymbol{H}}$. In the PN case, we propose a method for evaluating the matrix-vector product $\boldsymbol{L}_{\boldsymbol{H}}\boldsymbol{v}$, this is covered in Algorithm 3.

Complexity of $\boldsymbol{L}_{\boldsymbol{H}}\boldsymbol{v}$ evaluation Algorithm 3 is governed by an outer loop which performs $N-1$ iterations, see line 5. Inside the loop, the most expensive operations are in lines 6, 9, 10, 11 and 12, the rest of operations can be performed in

Algorithm 1 Branch and Bound, adapted from (Bunel et al., 2020a)

function BAB($\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma$)
  $\mathtt{global\_ub} \leftarrow \inf$
  $\mathtt{global\_lb} \leftarrow -\inf$
  $\alpha \leftarrow \mathtt{compute\_alpha}(\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma)$ ▷ Algorithm 2
  $\mathtt{probs} \leftarrow [(\mathtt{global\_lb}, \mathtt{l}, \mathtt{u})]$
  while $\mathtt{global\_ub} - \mathtt{global\_lb} > 10^{-6}$ **and** $\mathtt{global\_lb} \leq 0$ **do**
    $([], \mathtt{l}', \mathtt{u}') \leftarrow \mathtt{pick\_out}(\mathtt{probs})$ ▷ Take subset with the minimum lower bound
    $[(\mathtt{l\_1}, \mathtt{u\_1}), \ldots, (\mathtt{l\_s}, \mathtt{u\_s})] \leftarrow \mathtt{split}(\mathtt{l}', \mathtt{u}')$ ▷ Split widest input variable interval in halves
    for $i = 1 \ldots s$ **do**
      $\mathtt{prob\_ub} \leftarrow \mathtt{compute\_UB}(\mathtt{f}, \mathtt{l\_i}, \mathtt{u\_i}, \mathtt{t}, \gamma)$ ▷ PGD over the original function $g(\boldsymbol{z})$
      $\mathtt{prob\_lb} \leftarrow \mathtt{compute\_LB}(\mathtt{f}, \mathtt{l\_i}, \mathtt{u\_i}, \mathtt{t}, \gamma, \alpha)$ ▷ PGD over $g_\alpha(\boldsymbol{z}, \alpha)$
      if $\mathtt{prob\_ub} < \mathtt{global\_ub}$ **then**
        $\mathtt{global\_ub} \leftarrow \mathtt{prob\_ub}$
        $\mathtt{prune\_problems}(\mathtt{probs}, \mathtt{global\_ub})$ ▷ Remove if $\mathtt{prob\_lb} > \mathtt{global\_ub}$
      end if
      if $\mathtt{prob\_lb} < \mathtt{global\_ub}$ **and** $\mathtt{prob\_lb} \leq 0$ **then**
        $\mathtt{probs.append}((\mathtt{prob\_lb}, \mathtt{l\_i}, \mathtt{u\_i}))$
      end if
      if $\mathtt{prob\_ub} \leq 0$ **then** ▷ An adversarial example was found
        return $[], 0$
      end if
    end for
    if $|\mathtt{probs}| == 0$ **then** ▷ $\mathtt{prob\_lb} > 0$ for every subset
      return $[], 1$
    end if
    $\mathtt{global\_lb} \leftarrow \min\{\mathtt{lb} \mid (\mathtt{lb}, [], []) \in \mathtt{probs}\}$
  end while
  return $\mathtt{global\_ub}, \mathtt{global\_ub} > 0$
end function

Algorithm 2 Power method for α estimation

function COMPUTE_ALPHA($\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma$)
  $\mathtt{v} \leftarrow \mathtt{init\_v}(\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma)$ ▷ Ensure v is not an eigenvector of $\boldsymbol{L_H}$ and $||\mathtt{v}||_2 = 1$
  $\mathtt{prev\_v} \leftarrow \mathbf{0}$
  $\mathtt{r} \leftarrow 0$
  while $||\mathtt{v} - \mathtt{prev\_v}||_2 > 10^{-6}$ **do**
    $\mathtt{prev\_v} \leftarrow \mathtt{v}$
    $\mathtt{v} \leftarrow \mathtt{evaluate\_LHv}(\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma, \mathtt{v})$ ▷ Algorithm 3
    $\mathtt{r} \leftarrow ||\mathtt{v}||_2$
    $\mathtt{v} \leftarrow \mathtt{v}/\mathtt{r}$
    $\mathtt{v} \leftarrow \mathtt{evaluate\_LHv}(\mathtt{f}, \mathtt{l}, \mathtt{u}, \mathtt{t}, \gamma, \mathtt{v})$ ▷ Evaluate $\boldsymbol{L_H v}$ twice to deal with negative eigenvalues.
    $\mathtt{r} \leftarrow ||\mathtt{v}||_2$
    $\mathtt{v} \leftarrow \mathtt{v}/\mathtt{r}$
  end while
  return $\mathtt{r}/2$
end function

$O(d)$ time. In line 5, the bounds of the gradients are computed, this operation can be performed for every level $n \in [N]$ outside the main loop and store the results using $O(N \cdot d \cdot k)$ time. For lines 9, 10, 11 and 12, an outer loop with k iterations is used for the summation. Then, inside the summation, four dot products plus vector-scalar multiplications are performed, leading to a time complexity of $O(k \cdot d)$. Overall, the complexity of Algorithm 3 is $O(N \cdot d \cdot k)$.

Algorithm 3 Evaluation of $\boldsymbol{L_H v}$ for a CCP PN, implementation of Proposition 1

1: **function** EVALUATE_LHV(f, problem, v, t, γ)
2: $\quad$ lw $\leftarrow \boldsymbol{c}_{t:} - \boldsymbol{c}_{\gamma:}$
3: $\quad$ uw $\leftarrow \boldsymbol{c}_{t:} - \boldsymbol{c}_{\gamma:}$ ▷ Initial upper and lower bounds of the δ weight are given by the last linear layer.
4: $\quad$ result $\leftarrow \mathbf{0}$
5: $\quad$ **for** $n =$ degree$(f) \dots 2$ **do**
6: $\quad\quad$ lg, ug $\leftarrow \mathcal{L}(\boldsymbol{J_z}(\boldsymbol{x}^{(n)})), \mathcal{U}(\boldsymbol{J_z}(\boldsymbol{x}^{(n)}))$
7: $\quad\quad$ S1 $\leftarrow$ {lg · lw, lg · uw, ug · lw, ug · uw}
8: $\quad\quad$ lg, ug $\leftarrow$ min(S1), max(S1)
9: $\quad\quad$ Lv $\leftarrow \sum_{i=1}^{k} \boldsymbol{w}_{[n]:i}{}^{+} \cdot$ lg[i, :]$^\top$ v $+ \boldsymbol{w}_{[n]:i}{}^{-} \cdot$ ug[i, :]$^\top$ v $+$ lg[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{+}$ v $+$ ug[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{-}$ v
10: $\quad\quad$ Uv $\leftarrow \sum_{i=1}^{k} \boldsymbol{w}_{[n]:i}{}^{+} \cdot$ ug[i, :]$^\top$ v $+ \boldsymbol{w}_{[n]:i}{}^{-} \cdot$ lg[i, :]$^\top$ v $+$ ug[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{+}$ v $+$ lg[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{-}$ v
11: $\quad\quad$ L1 $\leftarrow \sum_{i=1}^{k} \boldsymbol{w}_{[n]:i}{}^{+} \cdot$ lg[i, :]$^\top \mathbf{1} + \boldsymbol{w}_{[n]:i}{}^{-} \cdot$ ug[i, :]$^\top \mathbf{1} +$ lg[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{+} \mathbf{1} +$ ug[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{-} \mathbf{1}$
12: $\quad\quad$ U1 $\leftarrow \sum_{i=1}^{k} \boldsymbol{w}_{[n]:i}{}^{+} \cdot$ ug[i, :]$^\top \mathbf{1} + \boldsymbol{w}_{[n]:i}{}^{-} \cdot$ lg[i, :]$^\top \mathbf{1} +$ ug[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{+} \mathbf{1} +$ lg[i, :] $\cdot \boldsymbol{w}_{[n]:i}^\top{}^{-} \mathbf{1}$
13: $\quad\quad$ result $\leftarrow$ result $+$ (Lv $+$ Uv)/2 $+$ ((L1 $-$ U1)/2) $*$ v
14: $\quad\quad$ lx, ux $\leftarrow \mathcal{L}(\boldsymbol{W}_{[n]}^\top \boldsymbol{z} + 1), \mathcal{U}(\boldsymbol{W}_{[n]}^\top \boldsymbol{z} + 1)$
15: $\quad\quad$ S2 $\leftarrow$ {lx · lw, lx · uw, ux · lw, ux · uw}
16: $\quad\quad$ lw, uw $\leftarrow$ min(S2), max(S2) ▷ Update bounds of the weight ($\mathcal{L}\,\delta', \mathcal{U}\,\delta'$), see Proposition 1
17: $\quad$ **end for**
18: $\quad$ **return** result
19: **end function**

D. Proofs

Proposition 1. *Let δ be a real-valued upper and lower bounded weight, the matrix-vector products $\mathcal{L}(\delta \cdot \nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v}$ and $\mathcal{U}(\delta \cdot \nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v}$ can be evaluated as:*

$$\begin{aligned}
\mathcal{L}(\delta \cdot \nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v} = {} & \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top{}^{+} \boldsymbol{v} \\
& + \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top{}^{-} \boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{+} \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)^\top}) \boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{-} \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)^\top}) \boldsymbol{v} \\
& + \mathcal{L}(\delta' \nabla^2_{\boldsymbol{zz}} x_i^{(n-1)}) \boldsymbol{v}
\end{aligned} \tag{36}$$

$$\begin{aligned}
\mathcal{U}(\delta \cdot \nabla^2_{\boldsymbol{zz}} x_i^{(n)})\boldsymbol{v} = {} & \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top{}^{-} \boldsymbol{v} \\
& + \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top{}^{+} \boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{-} \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)^\top}) \boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{+} \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)^\top}) \boldsymbol{v} \\
& + \mathcal{U}(\delta' \nabla^2_{\boldsymbol{zz}} x_i^{(n-1)}) \boldsymbol{v}\,,
\end{aligned} \tag{37}$$

where $\delta' = \delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1)$ and vectors $\mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})$ and $\mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})$ can be obtained through IBP on Eq. (8).

D.1. Convergence of the BaB algorithm to the global minima.

In this section we demonstrate a key property for verification: convergence to the global minima of Eq. (4). Let us firstly define some concepts of the BaB algorithm.

Let S_k be the subset picked at iteration k of the BaB algorithm and $\{S_{kq}|q=0,1,...\}$ be the sequence of recursive subsets, so that $S_{kq} \subset S_{kq-1}$ and $S_{k0} = S_k$. Let $\mathcal{L}(S_{kq})$ be the lower bound of subset S_{kq} and $\mathcal{L}_{kq} = \min\{\mathcal{L}(S_{kq})|q=0,1,...\}$ the lower bound in the branch rooted by subset S_k, we analogously define $\mathcal{U}(S_{kq})$ and $\mathcal{U}_{kq}$. Finally, let a **fathomed** or **pruned** set S_{kq} be a set where $\mathcal{L}(S_{kq}) > \mathcal{U}_{kq}$. From Horst & Tuy (1996, Definition IV.4):

Definition 1. *A bounding operation is called **consistent** if at every step any unfathomed subset can be further split, and if any infinitely decreasing sequence $\{S_{kq}|q=0,1,...\}$ for successively refined partition elements satisfies:*

$$\lim_{q\to\infty} (\mathcal{U}_{kq} - \mathcal{L}(S_{kq})) = 0\,. \tag{38}$$

Remark. Because $\mathcal{U}(S_{kq}) \geq \mathcal{U}_{kq} \geq \mathcal{L}(S_{kq})$, it suffices to prove that $\lim_{q\to\infty}(\mathcal{U}(S_{kq}) - \mathcal{L}(S_{kq}))$ to show a bounding operation is consistent.

Another relevant property is the subset selection being **bound improving**. Let $\mathcal{P}_k$ be the set of unfathomed subsets at iteration k (`probs` variable in Algorithm 1), from Horst & Tuy (1996, Definition IV.6):

Definition 2. *A subset selection operation is called **bound improving** if, at least each time after a finite number of steps, S_k satisfies the relation:*

$$S_k = \arg\min\{\mathcal{L}(S) : S \in \mathcal{P}_k\}\,. \tag{39}$$

Then, this ensures at least one partition element where the actual lower bound is attained is selected for further partition in step k of the algorithm.

Finally, Horst & Tuy (1996, Theorem IV.3) cover global convergence of general BaB algorithms.

Theorem 1. *In a BB procedure, suppose that the bounding operation is consistent and the subset selection operation is bound improving. Then the procedure is convergent:*

$$\mathcal{L} \coloneqq \lim_{k\to\infty} \mathcal{L}_k = \lim_{k\to\infty} f(\boldsymbol{z}^{(k)}) = \min_{\boldsymbol{z}\in C_{in}} f(\boldsymbol{z}) = \lim_{k\to\infty} \mathcal{U}_k = \mathcal{U}\,, \tag{40}$$

where C_{in} is the feasible set of the initial problem, $\mathcal{L}_k$ and $\mathcal{U}_k$ are the global lower and upper bounds at iteration k and $\boldsymbol{z}^{(k)}$ is the point where the upper bound $\mathcal{U}_k$ is attained.

Remark. In Theorem 1, $\mathcal{L}_k$ and $\mathcal{U}_k$ refer to variables `global_lb` and `global_ub` respectively in Algorithm 1.

Lemma 1. *Selecting the subset with the lowest lower bound at every iteration of a BaB algorithm is a bound improving subset selection strategy.*

Proof. By definition, when selecting the subset with the lowest lower bound, we are selecting at every iteration $S_k = \arg\min\{\mathcal{L}(S) : S \in \mathcal{P}_k\}$, which means that Eq. (39) holds at every iteration k and the strategy is bound improving. □

Definition 3. *Let subset S_k, $\{S_{kq}|q=0,1,...\}$ the sequence of recursive subsets rooted at $S_{k0} = S_k$ a branching operation is convergent iff $\lim_{q\to\infty}|S_{kq}| = 0$.*

Lemma 2. *Selecting the widest interval index $i = \arg\max \boldsymbol{u} - \boldsymbol{l}$ to split a problem, is a convergent branching operation.*

Proof. Supose at subset S_{kq}, we have bounds $\boldsymbol{l}^{(q)}$ and $\boldsymbol{u}^{(q)}$ and indexes $i_1, i_2, ..., i_d$ so that $u_{i_1}^{(q)} - l_{i_1}^{(q)} \geq u_{i_2}^{(q)} - l_{i_2}^{(q)} \geq \cdots \geq u_{i_d}^{(q)} - l_{i_d}^{(q)}$ is the decreasing ordered list of interval widths, then $|S_{kq}| = ||\boldsymbol{u}^{(q)} - \boldsymbol{l}^{(q)}||_2 = \sqrt{\sum_{j=1}^{d}(u_j^{(q)} - l_j^{(q)})^2} \leq \sqrt{\sum_{j=1}^{d}(u_{i_1}^{(q)} - l_{i_1}^{(q)})^2} = \sqrt{d(u_{i_1}^{(q)} - l_{i_1}^{(q)})^2} = (u_{i_1}^{(q)} - l_{i_1}^{(q)})\sqrt{d}$. Then, at subset S_{kq+1}, with bounds $\boldsymbol{l}^{(q+1)}$ and $\boldsymbol{u}^{(q+1)}$ and new indices $j_1, j_2, ..., j_d$, $u_{i_1}^{(q+1)} - l_{i_1}^{(q+1)} = \frac{u_{i_1}^{(q)} - l_{i_1}^{(q)}}{2}$ and then j_1 is either equal to i_1 or to i_2, depending on whether $\frac{u_{i_1}^{(q)} - l_{i_1}^{(q)}}{2} > u_{i_2}^{(q)} - l_{i_2}^{(q)}$ or not. Finally, as:

$$\begin{cases} u_{i_1}^{(q)} - l_{i_1}^{(q)} > \frac{u_{i_1}^{(q)} - l_{i_1}^{(q)}}{2} = u_{j_1}^{(q+1)} - l_{j_1}^{(q+1)} & \text{if } \frac{u_{i_1}^{(q)} - l_{i_1}^{(q)}}{2} > u_{i_2}^{(q)} - l_{i_2}^{(q)} \\ u_{i_1}^{(q)} - l_{i_1}^{(q)} > u_{i_2}^{(q)} - l_{i_2}^{(q)} = u_{j_1}^{(q+1)} - l_{j_1}^{(q+1)} & \text{if } \frac{u_{i_1}^{(q)} - l_{i_1}^{(q)}}{2} \leq u_{i_2}^{(q)} - l_{i_2}^{(q)} \end{cases} \tag{41}$$

and $u_{i_1}^{(q)} - l_{i_1}^{(q)} \geq 0$, the sequence $\{u_{i_1}^{(q)} - l_{i_1}^{(q)}\}_q$ is strictly decreasing and lower bounded by 0, then $\lim_{q\to\infty} u_{i_1}^{(q)} - l_{i_1}^{(q)} = 0$ must hold. Then $\lim_{q\to 0}(u_{i_1}^{(q)} - l_{i_1}^{(q)})\sqrt{d} = 0$ and as $(u_{i_1}^{(q)} - l_{i_1}^{(q)})\sqrt{d} \geq |S_{kq}| \geq 0$, the property $\lim_{q\to\infty} |S_{kq}| = 0$ holds and by Definition 3, the branching operation is convergent. □

A consequence of Lemma 2 is the following Corollary.

Corollary 2. *For any $\boldsymbol{z} \in [\boldsymbol{l}^{(q)}, \boldsymbol{u}^{(q)}]$, if $\lim_{q\to\infty} |S_{kq}| = 0$, in the limit $l_i^{(q)} = z_i = u_i^{(q)}\ \forall i = 1, ..., d, \forall \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$.*

Lemma 3. *Let widest interval selection be the branching operation, lower bounds obtained by α-convexification with $\alpha \geq \max\{0, -\frac{1}{2}\min\{\lambda_{min}(\boldsymbol{H}_f(\boldsymbol{z})) : \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\}\}$ and upper bounds obtained by evaluating $\mathcal{U}(S_{kq}) = g(\boldsymbol{z}^{upper})$ for any $\boldsymbol{z}^{upper} \in [\boldsymbol{l}, \boldsymbol{u}]$ are consistent.*

Proof. By Definition 1 is sufficient to check that $\lim_{q\to\infty} (\mathcal{U}(S_{kq}) - \mathcal{L}(S_{kq})) = 0$. By definition, the lower bound $\mathcal{L}(S_{kq}))$ is the solution to the function in Eq. (5) subject to $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$, which will lead to an optimal $\boldsymbol{z}^{\text{opt}}$ and $\mathcal{L}(S_{kq})) = g_\alpha(\boldsymbol{z}^{\text{opt}})$. If the upper bound is given by evaluating the objective function at any point $\boldsymbol{z}^{\text{upper}} \in [\boldsymbol{l}, \boldsymbol{u}]$ e.g., $\boldsymbol{z}^{\text{upper}} = \boldsymbol{z}^{\text{opt}}$ or in our case $\boldsymbol{z}^{\text{upper}} = \boldsymbol{z}^{\text{PGD}}$, the point obtained by performing PGD over g, $\mathcal{U}(S_{kq}) = f(\boldsymbol{z}^{\text{upper}})$, and their difference is: $\mathcal{U}(S_{kq}) - \mathcal{L}(S_{kq}) = g(\boldsymbol{z}^{\text{upper}}) - g(\boldsymbol{z}^{\text{opt}}) - \alpha\sum_{i=1}^{d}(z_i^{\text{opt}} - l_i^{(q)})(z_i^{\text{opt}} - u_i^{(q)})$. By virtue of Theorem 2, in the limit, $\boldsymbol{l}^{(q)} = \boldsymbol{z}^{\text{opt}} = \boldsymbol{z}^{\text{upper}} = \boldsymbol{u}^{(q)}$ and therefore $\lim_{q\to\infty} \mathcal{U}(S_{kq}) - \mathcal{L}(S_{kq}) = g(\boldsymbol{l}) - g(\boldsymbol{l}) - \alpha\sum_{i=1}^{d}(l_i^{(q)} - l_i^{(q)})(l_i^{(q)} - l_i^{(q)}) = 0$ and the bounds are consistent. □

Lemma 4. *Let widest interval selection be the branching operation, lower bounds obtained by IBP and upper bounds obtained by evaluating $\mathcal{U}(S_{kq}) = g(\boldsymbol{z}^{upper})$ for any $\boldsymbol{z}^{upper} \in [\boldsymbol{l}, \boldsymbol{u}]$ are consistent.*

Proof. By Definition 1 is sufficient to check that $\lim_{q\to\infty} (\mathcal{U}(S_{kq}) - \mathcal{L}(S_{kq})) = 0$. By definition, the lower bound $\mathcal{L}(S_{kq})$ is given by $\mathcal{L}(g(\boldsymbol{z})) = \mathcal{L}(f(\boldsymbol{z})_t) - \mathcal{U}(f(\boldsymbol{z})_\gamma)$, see Section 3.1, and the upper bound is given by $\mathcal{U}(S_{kq})) = g(\boldsymbol{z}^{\text{upper}})$. By virtue of Theorem 2, in the limit, $\boldsymbol{l}^{(q)} = \boldsymbol{z}^{\text{upper}} = \boldsymbol{u}^{(q)}$, then $\lim_{q\to\infty} \mathcal{U}(S_{kq})) = g(\boldsymbol{l})$. Then, for Eq. (22), it can be easily found that $\lim_{q\to\infty} \mathcal{L}(\hat{\boldsymbol{x}}^{(n)}) = \lim_{q\to\infty} \mathcal{U}(\hat{\boldsymbol{x}}^{(n)}) = \boldsymbol{W}_{[n]}^{\top}\boldsymbol{l}$. Then, for Eq. (23), every element in the sets S will converge to the same value and therefore, $\mathcal{L}(\boldsymbol{x}^{(n)}) = \min S = \max S = \mathcal{U}(\boldsymbol{x}^{(n)})$ for both CCP and NCP for every layer n. Finally, for the network's output, because $\mathcal{L}(x_i^{(N)}) = \mathcal{U}(x_i^{(N)})$, then, from Eq. (24), $\mathcal{L}(f(\boldsymbol{z})_i) = \mathcal{U}(f(\boldsymbol{z})_i) = f(\boldsymbol{l})_i, \forall i \in [o]$. Then, $\mathcal{L}(S_{kq})) = \mathcal{L}(f(\boldsymbol{z})_t - f(\boldsymbol{z})_\gamma) = \mathcal{L}(f(\boldsymbol{z})_t) - \mathcal{U}(f(\boldsymbol{z})_\gamma) = f(\boldsymbol{l})_t - f(\boldsymbol{l})_\gamma = f(\boldsymbol{z}^{\text{upper}})_t - f(\boldsymbol{z}^{\text{upper}})_\gamma = \mathcal{U}(S_{kq}))$ and the property holds.

□

A consequence of Lemmas 2 to 4 is:

Corollary 3. *Any branch and bound procedure with widest interval selection for branching, lowest lower bound subset selection and a bounding operation of: IBP or α-convexification, is convergent.*

Proof. Because of Lemma 2, we have convergence of the branching procedure. Then, due to Lemmas 3 and 4, we have bound consistency for both IBP and α-convexification bounding mechanisms. By Lemma 1, we have that the subset selection strategy is bound improving. Therefore, because of Theorem Theorem 1, we have that any BaB algorithm with these properties converges to a global minimizer. □

D.2. Lower bound of the minimum eigenvalue of the Hessian of PNs

Proof of Proposition 1. Let $\delta' = \delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1)$. Using the IBP rules from Section 3.1 and Appendix B.5, we can develop:

$$
\begin{aligned}
\mathcal{L}(\delta \cdot \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)}) &= \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top + \{\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top\}^\top \\
&\quad + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)})) && \text{[Eq. (9)]} \\
&= \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top) + \mathcal{L}(\boldsymbol{w}_{[n]:i} \delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{L}(\delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Distributive]} \\
&= \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^+}^\top + \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^-}^\top \\
&\quad + \boldsymbol{w}_{[n]:i}^+ \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) + \boldsymbol{w}_{[n]:i}^- \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{L}(\delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Linearity Eq. (6)]} \\
&= \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^+}^\top + \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^-}^\top \\
&\quad + \boldsymbol{w}_{[n]:i}^+ \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) + \boldsymbol{w}_{[n]:i}^- \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{L}(\delta' \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Definition of } \delta'\text{]}
\end{aligned}
\tag{42}
$$

Analogously, for the upper bound:

$$
\begin{aligned}
\mathcal{U}(\delta \cdot \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)}) &= \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top + \{\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top\}^\top \\
&\quad + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)})) && \text{[Eq. (9)]} \\
&= \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)}) \boldsymbol{w}_{[n]:i}^\top) + \mathcal{U}(\boldsymbol{w}_{[n]:i} \delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{U}(\delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Distributive]} \\
&= \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^+}^\top + \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^-}^\top \\
&\quad + \boldsymbol{w}_{[n]:i}^+ \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) + \boldsymbol{w}_{[n]:i}^- \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{U}(\delta \cdot (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Linearity Eq. (6)]} \\
&= \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^+}^\top + \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})) {\boldsymbol{w}_{[n]:i}^-}^\top \\
&\quad + \boldsymbol{w}_{[n]:i}^+ \mathcal{U}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) + \boldsymbol{w}_{[n]:i}^- \mathcal{L}(\delta \cdot (\nabla_{\boldsymbol{z}} x_i^{(n-1)})^\top) \\
&\quad + \mathcal{U}(\delta' \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}) && \text{[Definition of } \delta'\text{]}\,.
\end{aligned}
\tag{43}
$$

□

D.3. Lower bound of the Hessian's minimum eigenvalue for the product of polynomials case.

To verify a product of polynomials network, we need a lower bound of the minimum eigenvalue of its Hessian. In Proposition 2 we propose a valid lower bound.

Proposition 2. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the input and output of a polynomial. Let*

$$
\hat{\boldsymbol{J}}_{\boldsymbol{z}}(\boldsymbol{x}) = \arg\max\{\rho(\boldsymbol{J}\boldsymbol{J}^\top) : \boldsymbol{J} \in [\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})), \mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}))]\} = \max\{|\,\mathcal{L}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}))|, |\,\mathcal{U}(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x}))|\} \tag{44}
$$

be the Jacobian matrix with the largest possible norm. Let ρ be the spectral radius of a matrix. For all $\boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]$, the minimum eigenvalue of the hessian matrix of every position i ($\lambda_{min}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} y_i)$) satisfies:

$$
\lambda_{min}\Big(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} y_i\Big) \geq \sum_{j=i}^{k} \lambda_{min}\Big(\frac{\partial y_i}{\partial x_j} \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_j\Big) - \rho\Big(\hat{\boldsymbol{J}}_{\boldsymbol{z}}(\boldsymbol{x}) \hat{\boldsymbol{J}}_{\boldsymbol{z}}^\top(\boldsymbol{x})\Big) \cdot \rho\Big(\boldsymbol{L}_{\boldsymbol{H}}(\nabla^2_{\boldsymbol{x}\boldsymbol{x}} y_i)\Big)\,. \tag{45}
$$

Table 3: Description of convolutional PNs used in our experiments.

Name	degree / N	kernel size	stride	padding	channels
PN_Conv2	2	5×5	2	2	32
PN_Conv4	4	7×7	4	3	64

Proof.

$$
\begin{aligned}
\lambda_{\min}\Big(\nabla^2_{\boldsymbol{zz}} y_i\Big) &= \lambda_{\min}\Big(\sum_{j=i}^{k} \frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j + \boldsymbol{J}_{\boldsymbol{z}}^\top(\boldsymbol{x})\nabla^2_{\boldsymbol{xx}} y_i \boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\Big) && \text{[Eq. (20)]} \\
&\geq \lambda_{\min}\Big(\sum_{j=i}^{k} \frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j\Big) + \lambda_{\min}\Big(\boldsymbol{J}_{\boldsymbol{z}}^\top(\boldsymbol{x})\nabla^2_{\boldsymbol{xx}} y_i \boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\Big) && \text{[Weyl's inequality]} \\
&\geq \sum_{j=i}^{k} \lambda_{\min}\Big(\frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j\Big) + \lambda_{\min}\Big(\boldsymbol{J}_{\boldsymbol{z}}^\top(\boldsymbol{x})\boldsymbol{L}_{\boldsymbol{H}}(\nabla^2_{\boldsymbol{xx}} y_i)\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\Big) && \text{[Eq. (11)]} \\
&\geq \sum_{j=i}^{k} \lambda_{\min}\Big(\frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j\Big) - \rho\Big(\boldsymbol{J}_{\boldsymbol{z}}^\top(\boldsymbol{x})\boldsymbol{L}_{\boldsymbol{H}}(\nabla^2_{\boldsymbol{xx}} y_i)\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\Big) && \text{[Definition of } \rho\text{]} \\
&\geq \sum_{j=i}^{k} \lambda_{\min}\Big(\frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j\Big) - \rho(\boldsymbol{J}_{\boldsymbol{z}}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{z}}^\top(\boldsymbol{x}))\rho\Big(\boldsymbol{L}_{\boldsymbol{H}}(\nabla^2_{\boldsymbol{xx}} y_i)\Big) \\
&\geq \sum_{j=i}^{k} \lambda_{\min}\Big(\frac{\partial y_i}{\partial x_j}\nabla^2_{\boldsymbol{zz}} x_j\Big) - \rho\Big(\hat{\boldsymbol{J}}_{\boldsymbol{z}}(\boldsymbol{x})\hat{\boldsymbol{J}}_{\boldsymbol{z}}^\top(\boldsymbol{x}))\rho(\boldsymbol{L}_{\boldsymbol{H}}(\nabla^2_{\boldsymbol{xx}} y_i)\Big), && \text{[Eq. (44)]}
\end{aligned}
\tag{46}
$$

where in the second to last inequality we use $\rho(\boldsymbol{B}^\top\boldsymbol{A}\boldsymbol{B}) = \rho(\boldsymbol{B}\boldsymbol{B}^\top\boldsymbol{A}) \leq \rho(\boldsymbol{B}\boldsymbol{B}^\top)\rho(\boldsymbol{A}), \forall \boldsymbol{A} \in \mathbb{R}^{d_1 \times d_1}, \boldsymbol{B} \in \mathbb{R}^{d_1 \times d_2}$. □

E. Auxiliary experimental results and discussion

We start this section by comparing the performance of our method with ReLU NN verification algorithms Appendix E.1. Then, in Appendix E.2 we perform an ablation study on the effect the input size of the network has in our PN verification algorithm.

Additional notation: In addition to the notation already defined in the main paper, we use $\circ$ for convolutions.

E.1. Comparison with ReLU BaB verification algorithms.

Complete ReLU NN verification algorithms are usually benchmarked against other methods in the networks and ϵ values proposed by Singh et al. (2019). ReLU NN verification algorithms mostly rely on the specific structure of the networks, e.g. ReLU activation, which makes a direct comparison with ReLU nets hard. However, we match these benchmarks by training PNs with same degree as the number of ReLU layers and similar number of parameters. For a detailed description of these networks check Table 5 and Table 6.

When comparing with the SOTA verifier β-CROWN (Wang et al., 2021), we firstly observe that the upper bound of verified accuracy is very similar between the NNs and PNs fully connected benchmarks. However, for convolutional PNs (C-PNs) the upper bound is much lower, even 0 for PN_Conv4 trained in the MNIST dataset, see Table 4. Secondly, we observe that the gap between the upper bound and verified accuracy (`U.B.` and `Ver.%` in Table 4) is small for β-CROWN. In our case, this gap is only small for the shallowest PN and smallest ϵ (5×81 with $\epsilon = 0.015$), obtaining 88.0% verified accuracy and outperforming β-CROWN with only 77.4%. For PN_Conv2 trained in CIFAR10 and evaluated at $\epsilon = 2/255$, our verified accuracy, 15.7%, is also really close to the upper bound 16.1%, but both are low in comparison with the β-CROWN equivalent.

Table 4: Verification results on our proposed PN verification benchmarks. We run our verification procedure over the first 1000 images of the test split of each dataset. `Time(s)` refers to the average running time per image when the verification was not timed out, i.e., we can verify or falsify the property in the given time limit, `Ver.%` is the verified accuracy and `U.B.` its upper bound. We use the same ϵ values as Wang et al. (2021). We observe that in comparison with β-CROWN, our method generally has a larger gap between verified accuracy and its upper bound.

Dataset	β-CROWN* (Wang et al., 2021)				VPN			
	Model	Time(s)	Ver.%	U.B.	Model	Time(s)	Ver.%	U.B.
MNIST	6×100	102	69.9	84.2	5×30	34	24.7	81.1
	6×200	86	77.4	90.1	5×81	60	88.0	91.1
	9×100	103	62.0	82.0	8×24	33	0.0	80.2
	9×200	95	73.5	91.1	8×70	94	0.6	91.6
	ConvSmall	7.0	72.7	73.2	PN_Conv2	3.0	3.5	12.1
	ConvBig	3.1	79.3	80.4	PN_Conv4	8.9	0.0	0.0
CIFAR10	ConvSmall	6.8	46.3	48.1	PN_Conv2	63.4	15.7	16.1
	ConvBig	15.3	51.6	55.0	PN_Conv4	161.4	9.4	16.3

*Note: We report numbers from Wang et al. (2021).

Table 5: Comparison of PNs to the fully connected ReLU network benchmarks proposed in Singh et al. (2019). `#Par.` refers to the number of parameters in the ReLU benchmark and our PNs respectively. We build PNs with same degree as number of non-linearities in their corresponding ReLU NN benchmark. We also adjust the hidden size so that the number of parameters is matched.

ReLU NN	#Par.	Acc.(%)	PN	#Par.	Acc.(%)
6×100	$119,910$	96.0	5×30	$117,910$	97.2
9×100	$150,210$	94.7	8×24	$150,778$	97.5
6×200	$319,810$	97.2	5×81	$318,340$	96.2
9×200	$440,410$	95.0	8×70	$439,750$	96.7

E.2. Effect of the input size in PN verification

In this experiment we evaluate the effect of the input size in the verification of a PN. In order to evaluate this, we train 3 different CCP networks over the STL10 dataset. Each model has been trained with STL10 images preprocessed with 3 different resizing factors. Every network is a CCP_4×25 trained with a learning rate of 10^{-4}.

As seen in Table 7, downsampling the input images results in a decrease in the accuracy (i.e., from 35.1% to 31.8% at 32×32 resolution). However, downsampling the input improves the robustness of the network. For all ϵ values we observe less successful adversarial attacks, i.e., higher upper bound of the verified accuracy (`U.B.`). In addition, the verification process is improved, we obtain higher values for verified accuracy (`Ver.%`), but also the gap with its upper bound is reduced. We believe this phenomenon is due to the networks learning more robust representations with a smaller input size (Raghunathan et al., 2020; Zhang et al., 2019).

Table 6: Comparison of convolutional PNs (C-PNs) to the Convolutional ReLU network benchmarks proposed in Singh et al. (2019). `#Par.` refers to the number of parameters in the ReLU benchmark and our PNs respectively. We build C-PNs with same number of convolutional layers as their corresponding ReLU NN benchmark.

Dataset	ReLU NN	#Par.	Acc.(%)	PN	#Par.	Acc.(%)
MNIST	Conv Small	$89,606$	98.0	PN_Conv2	$114,218$	98.0
MNIST	Conv Big	$1,974,762$	92.9	PN_Conv4	$834,762$	98.0
CIFAR10	Conv Small	$125,318$	63.0	PN_Conv2	$133,994$	57.2
CIFAR10	Conv Big	$2,466,858$	63.1	PN_Conv4	$845,514$	58.3

Table 7: Input size ablation study: CCP_4×25 networks are trained over STL10 with three different input sizes. Both Verified accuracy (`Ver.%`) and its upper bound (`U.B.`) increase when the input size is reduced for all the studied ϵ values.

Input size	Acc.%	ϵ	VPN		
			Time(s)	Ver.%	U.B.
$3 \times 96 \times 96$ (original)	35.1	$1/255$	47.4	4.8	10.6
		$2/255$	19.0	0.0	2.4
		$8/255$	21.5	0.0	0.0
$3 \times 64 \times 64$	35.6	$1/255$	52.1	11.7	12.6
		$2/255$	20.7	0.6	3.2
		$8/255$	12.0	0.0	0.0
$3 \times 32 \times 32$	31.8	$1/255$	14.7	**17.9**	**18.0**
		$2/255$	21.4	**7.2**	**8.9**
		$8/255$	16.2	0.0	**0.1**