Sound and Complete Verification of Polynomial Networks

Anonymous Author(s)
Affiliation
Address
email

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 and Panagakis, 2020] being state-of-the-art on large-scale face recognition[1].Unlike the conventional Neural Networks (NNs), where non-linearitiy is introduced with the use of activation functions [LeCun et al., 2015], PNs are able to 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 and 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

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

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

on α-convexification [Adjiman and Floudas, 1996], which is integrated into a Branch and Bound algorithm [Land and 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. To encourage the community to improve the verification of PNs, we intend to make our source code public upon the acceptance of the paper.

The proposed approach can practically verify PNs and that could theoretically be applied for sound and complete verification of any twice-differentiable network. Recent works showing twice-differentiability of ReLU NNs almost everywhere [Yao et al., 2018] also suggest that our approach could be used in the case of ReLU NNs, which we believe is an interesting avenue for future work.

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.

Roadmap: We provide the necessary background by introducing the PN architecture and formalizing the Robustness Verification problem in Section 2. Section 3 provides a *sound* and *complete* method called VPN to tackle PN verification problem. In Section 4, we discuss the related works in the context of our contributions. Section 5 is devoted to experimental validation. Additional experiments, details and proofs are deferred to the appendix.

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:

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

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. 1. 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

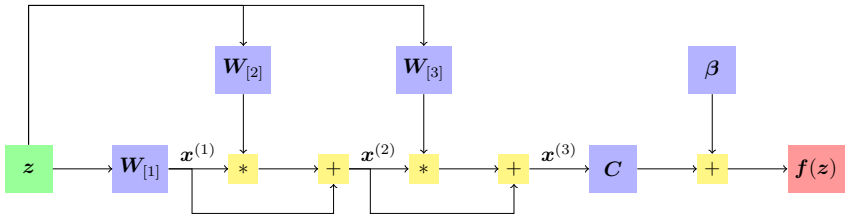

Figure 1: Third degree PN architecture. Blue boxes depict learnable parameters, yellow depict mathematical operations, the green and red boxes are the input and the output respectively. Note that no activation functions are involved, only element-wise (Hadamard) products $*$ and additions $+$. This figure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

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:

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

In this work we focus on *adversarial robustness* [Szegedy et al., 2014, Carlini and 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:

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

In other words, we need an algorithm that given a function f, an input $\boldsymbol{z}_0$ and an adversarial budget ϵ, checks whether Eq. (2) is satisfied. In the case of ReLU NNs, this has been proven to be an NP-complete problem [Katz et al., 2017]. 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:

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

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.

There are two main properties that we would like to have in a verification algorithm: *soundness* and *completeness*. An algorithm is *sound* (*complete*) if every time it verifies (falsifies) a property, it is guaranteed to be the correct answer. In practice, when an algorithm is guaranteed to provide the exact global minima of Eq. (4), i.e., v^*, it is said to be *sound* and *complete* (usually referred in the literature as simply *complete* [Ferrari et al., 2022]), whereas if a lower bound of it is provided $\hat{v}^* \leq v^*$, the algorithm is *sound* but not *complete*. We will not consider just *complete* verification, which simply aims at looking for adversarial examples, e.g., Madry et al. [2018]. For a deeper discussion on *soundness* and *completeness*, we refer to Liu et al. [2021].

3 Method

Our method, called VPN, can be categorized in the the Branch and Bound (BaB) framework [Land and Doig, 1960], a well known approach to global optimization [Horst and 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 and 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 function of any twice-differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ such that

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

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

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.

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:

$$\mathcal{L}(g(\boldsymbol{z})) \leq g(\boldsymbol{z}), \quad \mathcal{U}(g(\boldsymbol{z})) \geq g(\boldsymbol{z}), \forall \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}] \tag{6}$$

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:

$$\begin{aligned}
&\textbf{Identity} \begin{cases} \mathcal{L}(z_i) = l_i \\ \mathcal{U}(z_i) = u_i \end{cases} \\
&\textbf{linear mapping} \begin{cases} \mathcal{L}(\sum_i w_i h_i(\boldsymbol{z})) = w_i^+\, \mathcal{L}(h_i(\boldsymbol{z})) + w_i^-\, \mathcal{U}(h_i(\boldsymbol{z})) \\ \mathcal{U}(\sum_i w_i h_i(\boldsymbol{z})) = w_i^-\, \mathcal{L}(h_i(\boldsymbol{z})) + w_i^+\, \mathcal{U}(h_i(\boldsymbol{z})) \end{cases} \\
&\textbf{multiplication} \begin{cases} S = \begin{Bmatrix} \mathcal{L}(h_1(\boldsymbol{z}))\, \mathcal{L}(h_2(\boldsymbol{z})), \\ \mathcal{L}(h_1(\boldsymbol{z}))\, \mathcal{U}(h_2(\boldsymbol{z})), \\ \mathcal{U}(h_1(\boldsymbol{z}))\, \mathcal{L}(h_2(\boldsymbol{z})), \\ \mathcal{U}(h_1(\boldsymbol{z}))\, \mathcal{U}(h_2(\boldsymbol{z})) \end{Bmatrix}, |S| = 4 \\ \mathcal{L}(h_1(\boldsymbol{z}) h_2(\boldsymbol{z})) = \min S\,, \\ \mathcal{U}(h_1(\boldsymbol{z}) h_2(\boldsymbol{z})) = \max S\,, \end{cases}
\end{aligned} \tag{7}$$

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. Note that the set S is equivalent to:

$$S = \left\{ ab \middle|\, \forall a \in \{\mathcal{L}(h_1(\boldsymbol{z})), \mathcal{U}(h_1(\boldsymbol{z}))\}\,, \forall b \in \{\mathcal{L}(h_2(\boldsymbol{z})), \mathcal{U}(h_2(\boldsymbol{z}))\} \right\}\,.$$

With these basic operations one can define bounds on any intermediate output, gradient or Hessian of a PN, e.g., the lower bound on the recursive formula from Eq. (1) can be expressed as:

$$\mathcal{L}(x_i^{(n)}) = \mathcal{L}((\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z})x_i^{(n-1)} + x_i^{(n-1)}) = \mathcal{L}((\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z}+1)x_i^{(n-1)}),\ \forall i \in [k], n \in [N-1]+1\,, \quad (8)$$

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:

$$\mathcal{L}(\boldsymbol{g}(\boldsymbol{z})) = \begin{bmatrix} \mathcal{L}(g(\boldsymbol{z})_1) \\ \mathcal{L}(g(\boldsymbol{z})_2) \\ \vdots \\ \mathcal{L}(g(\boldsymbol{z})_m) \end{bmatrix}, \quad \mathcal{L}(\boldsymbol{G}(\boldsymbol{z})) = \begin{bmatrix} \mathcal{L}(g(\boldsymbol{z})_{11}) & \cdots & \mathcal{L}(g(\boldsymbol{z})_{1m_2}) \\ \vdots & \ddots & \\ \mathcal{L}(g(\boldsymbol{z})_{m_1 1}) & & \mathcal{L}(g(\boldsymbol{z})_{m_1 m_2}) \end{bmatrix}, \quad (9)$$

for $\boldsymbol{g}(\boldsymbol{z}) \in \mathbb{R}^m$ and $\boldsymbol{G}(\boldsymbol{z}) \in \mathbb{R}^{m_1 \times m_2}$. 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

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:

$$\nabla_{\boldsymbol{z}} g(\boldsymbol{z}) = \sum_{i=1}^{k} (c_{ti} - c_{\gamma i}) \nabla_{\boldsymbol{z}} x_i^{(N)}, \quad \boldsymbol{H}_g(\boldsymbol{z}) = \sum_{i=1}^{k} (c_{ti} - c_{\gamma i}) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)}\,, \quad (10)$$

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:

$$\nabla_{\boldsymbol{z}} x_i^{(n)} = \boldsymbol{w}_{[n]:i} \cdot x_i^{(n-1)} + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)} \quad (11)$$

$$\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)} = \nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top + \{\nabla_{\boldsymbol{z}} x_i^{(n-1)} \boldsymbol{w}_{[n]:i}^\top\}^\top + (\boldsymbol{w}_{[n]:i}^\top \boldsymbol{z} + 1) \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)}\,, \quad (12)$$

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.

In the next, 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:

$$\boldsymbol{L_H} = \frac{\mathcal{L}(\boldsymbol{M}) + \mathcal{U}(\boldsymbol{M})}{2} + \text{diag}\left(\frac{\mathcal{L}(\boldsymbol{M})\boldsymbol{1} - \mathcal{U}(\boldsymbol{A})\boldsymbol{1}}{2}\right), \quad (13)$$

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:

$$-\rho(\boldsymbol{L_H}) \leq \lambda_{\min}(\boldsymbol{L_H}) \leq \lambda_{\min}(\boldsymbol{H}_g(\boldsymbol{z})),\ \forall \boldsymbol{H}_g(\boldsymbol{z}) \in \mathcal{H},\ \ \boldsymbol{z} \in [\boldsymbol{l}, \boldsymbol{u}]\,, \quad (14)$$

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:

$$\boldsymbol{v}_n = \frac{\boldsymbol{M}(\boldsymbol{M}\boldsymbol{v}_{n-1})}{||\boldsymbol{M}(\boldsymbol{M}\boldsymbol{v}_{n-1})||_2}\,, \tag{15}$$

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 and Pollaczek-Geiringer, 1929].

Power method over lower bounding Hessian of PNs

We can employ IBP (Section 3.1) in order to obtain an expression of the lower bounding Hessian ($\boldsymbol{L_H}$) and evaluate Eq. (15) as:

$$\boldsymbol{L_H}\boldsymbol{v} = \frac{\mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} + \mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v}}{2} + \left(\frac{\mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\mathbf{1} - \mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\mathbf{1}}{2}\right) * \boldsymbol{v}\,. \tag{16}$$

Applying IBP on Eq. (10) we obtain:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} &= \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{+}\,\mathcal{L}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)})\boldsymbol{v} + \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{-}\,\mathcal{U}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)})\boldsymbol{v} \\
\mathcal{U}(\boldsymbol{H}_g(\boldsymbol{z}))\boldsymbol{v} &= \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{-}\,\mathcal{L}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)})\boldsymbol{v} + \sum_{i=1}^{k}(c_{ti} - c_{\gamma i})^{+}\,\mathcal{U}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(N)})\boldsymbol{v}\,.
\end{aligned} \tag{17}$$

We can recursively evaluate $\mathcal{L}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)})\boldsymbol{v}$ and $\mathcal{U}(\nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)})\boldsymbol{v}$ efficiently as these matrices can be expressed as a sum of rank-1 matrices:

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

$$\begin{aligned}
\mathcal{L}(\delta \cdot \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)})\boldsymbol{v} = {} & \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}{}^{+}\boldsymbol{v} + \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}{}^{-}\boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{+}\,\mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)\top})\boldsymbol{v} + \boldsymbol{w}_{[n]:i}{}^{-}\,\mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)\top})\boldsymbol{v} \\
& + \mathcal{L}(\delta' \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{v}
\end{aligned} \tag{18}$$

$$\begin{aligned}
\mathcal{U}(\delta \cdot \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n)})\boldsymbol{v} = {} & \mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}{}^{-}\boldsymbol{v} + \mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{w}_{[n]:i}^{\top}{}^{+}\boldsymbol{v} \\
& + \boldsymbol{w}_{[n]:i}{}^{-}\,\mathcal{L}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)\top})\boldsymbol{v} + \boldsymbol{w}_{[n]:i}{}^{+}\,\mathcal{U}(\delta \cdot \nabla_{\boldsymbol{z}} x_i^{(n-1)\top})\boldsymbol{v} \\
& + \mathcal{U}(\delta' \nabla^2_{\boldsymbol{z}\boldsymbol{z}} x_i^{(n-1)})\boldsymbol{v}\,,
\end{aligned} \tag{19}$$

where $\delta' \in [\mathcal{L}(\delta), \mathcal{U}(\delta)] \cdot [\mathcal{L}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z} + 1), \mathcal{U}(\boldsymbol{w}_{[n]:i}^{\top}\boldsymbol{z} + 1)]$ 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. (11)*.*

Lastly, by applying recursively Proposition 1 from $n = N$ to $n = 1$, starting with $\delta = 1$, we can substitute the results on Eq. (17) and then on Eq. (16) to efficiently evaluate a step of the power method (Eq. (15)) 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 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.

4.1 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.

4.2 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 and 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.

5 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 A. 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 run on a single-GPU machine.

Table 1: Verification results for 2^{nd} degree 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	Model	Correct	ϵ	VPN (Our method)				Gurobi			
				time	F	T	t.o.	time	F	T	t.o.
MNIST	2×16	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	2×16	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	2×16	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				

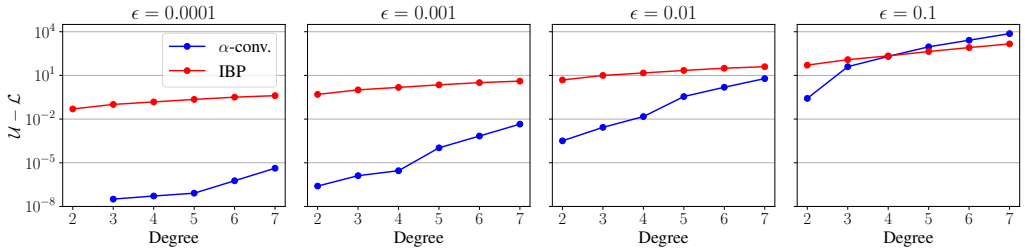

Figure 2: Average difference in log-scale between PGD upper bound ($\mathcal{U}$) and lower bound ($\mathcal{L}$) provided by α-convexification (blue) and IBP (red) of the first 1000 images of the MNIST dataset. α-convexification bounds are significantly tighter than IBP for small ϵ values and all PN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

5.1 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.

5.2 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 2^{nd} (PN_Conv2) and 4^{th} (PN_Conv4) PNs with both bounding methods and a maximum time of 120 seconds, for details on the architecture of these networks, we refer to Appendix A. When using IBP,

Table 2: Verification results with our method employing IBP and α-convexification for lower bounding the objective. `Acc.%` is the clean accuracy of the network, `Ver.%` is the verified accuracy and `U.B.` its upper bound. When using α-convexification bounds we get verified accuracies really close to the upper bound, while when using IBP verified accuracy is 0 for every network-ϵ pair, which makes it unsuitable for PN verification.

Dataset	Model	Acc.%	ϵ	IBP		VPN (α-convexification)		
				Time(s)	Ver.%	Time(s)	Ver.%	U.B.
MNIST	PN_Conv4	98.6	0.015	0.3	0.0	50	**96.3**	96.4
			0.026	0.4	0.0	69	**92.9**	94.8
CIFAR10	PN_Conv2	63.5	1/255	0.3	0.0	136.2	**44.4**	44.6
			2/255	0.5	0.0	89.2	**25.4**	27.5
	PN_Conv4	62.6	1/255	0.4	0.0	274.6	**45.5**	46.7
			2/255	0.5	0.0	224.1	**16.5**	30.5
STL10*	PN_Conv4	38.1	1/255	3.4	0.0	2481.0	**21.7**	21.9

*Note: Results obtained in the first 360 images of the dataset due to the longer running times because of the larger input size of STL10.

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 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 verified accuracy with IBP and α-convexification, see Table 2. With the latter, we are able to verify the accuracy of 2^{nd} and 4^{th} order PNs almost exactly (almost no gap between the verified accuracy and its upper bound) 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.

6 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.

Limitations: As discussed in Appendix A.2, our verification method does not scale to high-degree PNs. Even though we can verify high-accuracy PNs (see Table 2), we are still far from verifying the top performing deep PNs studied in Chrysos et al. [2021b]. Another problem that we share with ReLU NN verifiers is the scalability to networks with larger input size [Wang et al., 2021]. In this work we are able to verify networks trained in STL10 [Coates et al., 2011], but these networks are shallow, yet their verification still takes a long time, see Table 2.

Societal impact: The performance on standard image classification benchmarks has increased substantially the last few years, owing to the success of neural networks. Their success enables their adoption in tackling real-world problems. However, robustness and trustworthiness of neural networks is of critical importance before their adoption in real-world applications. Our method is a verifier that focuses on polynomial networks and enables the complete verification of PNs. Therefore, we expect that by using the proposed method, certain properties of the robustness could be verified in a principled way. We expect this to have a predominantly positive societal impact as either a tool for pre-trained models or tool for certifying models as part of their debugging. However, it can also be used as a tool to find weaknesses of pretrained PNs by adversarial agents.

References

Claire S. Adjiman and Christodoulos A. Floudas. 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.

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

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

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2621–2629, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. 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.

Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Lagrangian decomposition for neural network verification. *CoRR*, abs/2002.10410, 2020b. URL https://arxiv.org/abs/2002.10410.

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

Moulik Choraria, Leello Tadesse Dadi, Grigorios Chrysos, Julien Mairal, and Volkan Cevher. The spectral bias of polynomial neural networks. In *International Conference on Learning Representations (ICLR)*, 2022.

Grigorios Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. $\pi-$nets: Deep polynomial neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Grigorios G Chrysos and Yannis Panagakis. 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.

Grigorios G. Chrysos, Markos Georgopoulos, Jiankang Deng, and Yannis Panagakis. Polynomial networks in deep classifiers. *CoRR*, abs/2104.07916, 2021a. URL https://arxiv.org/abs/2104.07916.

Grigorios G. Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Jiankang Deng, Yannis Panagakis, and Stefanos P Zafeiriou. Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021b. ISSN 1939-3539. doi: 10.1109/tpami.2021.3058891. URL http://dx.doi.org/10.1109/TPAMI.2021.3058891.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/coates11a.html.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7714–7722, 2019.

Zehao Dou, Stanley J Osher, and Bao Wang. Mathematical analysis of adversarial attacks. *arXiv preprint arXiv:1811.06492*, 2018.

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

Fenglei Fan, Mengzhou Li, Fei Wang, Rongjie Lai, and Ge Wang. Expressivity and trainability of quadratic networks. *CoRR*, abs/2110.06081, 2021. URL https://arxiv.org/abs/2110.06081.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. 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.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *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.

Gaurav Goswami, Akshay Agarwal, Nalini K. Ratha, Richa Singh, and Mayank Vatsa. 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.

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

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks, 2017. URL https://arxiv.org/abs/1702.01135.

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

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

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

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

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

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

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

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. 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.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 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.

R. V. Mises and H. Pollaczek-Geiringer. 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.

Ramon E. Moore, Ralph Baker Kearfott, and Michael J. Cloud. Introduction to interval analysis. 2009.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *CoRR*, abs/2104.06718, 2021. URL https://arxiv.org/abs/2104.06718.

Vicenç Rúbies Royo, Roberto Calandra, Dušan M. Stipanović, and Claire J. Tomlin. Fast neural network verification via shadow prices. *ArXiv*, abs/1902.07247, 2019.

Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lWUoA9FQ.

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

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

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199.

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

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

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. 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.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018b. doi: 10.1109/CVPR.2018.00813.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4954–4964, Red Hook, NY, USA, 2018. Curran Associates Inc.

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

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Section 6 we discuss the limitations.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our method aims at verifying PNs robustness to adversarial attacks, which has no direct negative societal impact, see Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] The complete proof of propositions, lemmas and theorems is present in the appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We use publicly available datasets and our code and instructions are available in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] We follow the standard practice in the literature for reporting the results.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] A single GPU was used in each experiment. The single GPU is part of our internal cluster.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] Standard image-based datasets are used in this work. All of them are publicly available and we cite their respecting papers, where the licenses are mentioned.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]