DeepBern-Nets: Taming the Complexity of Certifying Neural Networks Using Bernstein Polynomial Activations and Precise Bound Propagation

Haitham Khedr, Yasser Shoukry

University of California, Irvine
{hkhedr, yshoukry}@uci.edu

Abstract

Formal certification of Neural Networks (NNs) is crucial for ensuring their safety, fairness, and robustness. Unfortunately, on the one hand, sound and complete certification algorithms of ReLU-based NNs do not scale to large-scale NNs. On the other hand, incomplete certification algorithms are easier to compute, but they result in loose bounds that deteriorate with the depth of NN, which diminishes their effectiveness. In this paper, we ask the following question; can we replace the ReLU activation function with one that opens the door to incomplete certification algorithms that are easy to compute but can produce tight bounds on the NN's outputs? We introduce DeepBern-Nets, a class of NNs with activation functions based on Bernstein polynomials instead of the commonly used ReLU activation. Bernstein polynomials are smooth and differentiable functions with desirable properties such as the so-called range enclosure and subdivision properties. We design a novel Interval Bound Propagation (IBP) algorithm, called Bern-IBP, to efficiently compute tight bounds on DeepBern-Nets outputs. Our approach leverages the properties of Bernstein polynomials to improve the tractability of neural network certification tasks while maintaining the accuracy of the trained networks. We conduct experiments in adversarial robustness and reachability analysis settings to assess the effectiveness of the approach. Our proposed framework achieves high certified accuracy for adversarially-trained NNs, which is often a challenging task for certifiers of ReLU-based NNs. This work establishes Bernstein polynomial activation as a promising alternative for improving NN certification tasks across various NNs applications.

1 Introduction

Deep neural networks (NNs) have revolutionized numerous fields with their remarkable performance on various tasks, ranging from computer vision and natural language processing to healthcare and robotics. As these networks become integral components of critical systems, ensuring their safety, security, fairness, and robustness is essential. It is unsurprising, then, the growing interest in the field of certified machine learning, which resulted in NNs with enhanced levels of robustness to adversarial inputs (Goodfellow, Shlens, and Szegedy 2014; Kurakin, Goodfellow, and Bengio 2016; Song et al. 2018; Szegedy et al. 2013), fairness (Zhang, Lemoine, and Mitchell 2018; Xu et al. 2018; Mehrabi et al. 2021; Khedr and Shoukry 2022), and correctness (Yang and Rinard 2019).

While certifying the robustness, fairness, and correctness of NNs with respect to formal properties is shown to be NP-hard (Katz et al. 2017), state-of-the-art certifiers rely on computing upper/lower bounds on the output of the NN and its intermediate layers (Wang et al. 2021; Khedr, Ferlez, and Shoukry 2021; Ferrari et al. 2022; Bak 2021; Henriksen and Lomuscio 2021). Accurate bounds can significantly reduce the complexity and computational effort required during the certification process, facilitating more efficient and dependable evaluations of the network's behavior in diverse and challenging scenarios. Moreover, computing such bounds has opened the door for a new set of "certified training" algorithms (Zhang et al. 2022; Lyu et al. 2021; Müller et al. 2022b) where these bounds are used as a regularizer that penalizes the worst-case violation of robustness or fairness, which leads to training NNs with favorable properties. While computing such lower/upper bounds is crucial, current techniques in computing lower/upper bounds on the NN outputs are either computationally efficient but result in loose lower/upper bounds or compute tight bounds but are computationally expensive. In this paper, we are interested in algorithms that can be both computationally efficient and lead to tight bounds.

This work follows a Design-for-Certifiability approach where we ask the question; can we replace the ReLU activation function with one that allows us to compute tight upper/lower bounds efficiently? Introducing such novel activation functions designed with certifiability in mind makes it possible to create NNs that are easier to analyze and certify during their training. Our contributions in this paper can be summarized as follows:

1. We introduce DeepBern-Nets, a NN architecture with a new activation function based on Bernstein polynomials. Our primary motivation is to shift some of the computational efforts from the certification phase to the training phase. By employing this approach, we can train NNs with known output (and intermediate) bounds for a predetermined input domain which can accelerate the certification process.
2. We present Bern-IBP, an Interval Bound Propagation (IBP) algorithm that computes tight bounds of

DeepBern-Nets leading to an efficient certifier.

3. We show that Bern-IBP can certify the adversarial robustness of adversarially-trained DeepBern-Nets on MNIST and CIFAR-10 datasets even with large architectures with millions of parameters. This is unlike state-of-the-art certifiers for ReLU networks, which often fail to certify robustness for adversarially-trained ReLU NNs.
4. We show that Bern-IBP can also be used for certified training, it achieves comparable results with other state-of-the-art certifiers in a controlled training setup as discussed in the SOK benchmark.

We believe that our framework, DeepBern-Nets and Bern-IBP, enables more reliable guarantees on NN behavior and contributes to the ongoing efforts to create safer and more secure NN-based systems, which is crucial for the broader deployment of deep learning in real-world applications.

2 DeepBern-Nets: Deep Bernstein Polynomial Networks

Bernstein Polynomials Preliminaries

Bernstein polynomials form a basis for the space of polynomials on a closed interval (Farouki 2012). These polynomials have been widely used in various fields, such as computer-aided geometric design (Farouki 2012), approximation theory (Qian, Riedel, and Rosenberg 2011), and numerical analysis (Farouki and Rajan 1987), due to their unique properties and intuitive representation of functions. A general polynomial of degree n in Bernstein form on the interval $[l, u]$ can be represented as:

$$P_n^{[l,u]}(x) = \sum_{k=0}^{n} c_k b_{n,k}^{[l,u]}(x), \qquad x \in [l, u] \tag{1}$$

where $c_k \in \mathbb{R}$ are the coefficients associated with the Bernstein basis $b_{n,k}^{[l,u]}(x)$, defined as:

$$b_{n,k}^{[l,u]}(x) = \frac{\binom{n}{k}}{(u-l)^n}(x-l)^k(u-x)^{n-k}, \tag{2}$$

with $\binom{n}{k}$ denoting the binomial coefficient. The Bernstein coefficients c_k determine the shape and properties of the polynomial $P_n^{[l,u]}(x)$ on the interval $[l, u]$. It is important to note that unlike polynomials represented in power basis form, the representation of a polynomial in Bernstein form depends on the domain of interest $[l, u]$ as shown in equation (1).

Networks with Bernstein Activation Functions

We propose using Bernstein polynomials as non-linear activation functions σ in feed-forward NNs. We call such NNs as DeepBern-Nets. Like feed-forward NNs, DeepBern-Nets consist of multiple layers, each consisting of linear weights followed by non-linear activation functions. Unlike conventional activation functions (e.g., ReLU, sigmoid, tanh, ..), Bernstein-based activation functions are parametrized with learnable Bernstein coefficients $\boldsymbol{c} = c_0, \ldots, c_n$, i.e.,

$$\sigma(x; l, u, \boldsymbol{c}) = \sum_{k=0}^{n} c_k b_{n,k}^{[l,u]}(x), \qquad x \in [l, u], \tag{3}$$

where x is the input to the neuron activation, and the polynomial degree n is an additional hyper-parameter of the Bernstein activation and can be chosen differently for each neuron. Figure 1 shows a simplified computational graph of the Bernstein activation and how it is used to replace conventional activation functions.

Training of DeepBern-Nets. Since Bernstein polynomials are defined on a specific domain (equation 2), we need to determine the lower and upper bounds ($\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$) of the inputs to the Bernstein activation neurons in layer k, during the training of the network. To that end, we assume that the input domain $\mathcal{D}$ is bounded with the lower and upper bounds (denoted as $\boldsymbol{l}^{(0)}$ and $\boldsymbol{u}^{(0)}$, respectively) known during training. We emphasize that our assumption that $\mathcal{D}$ is bounded and known is not conservative, as the input to the NN can always be normalized to $[0, 1]$, for example.

Using the bounds on the input domain $\boldsymbol{l}^{(0)}$ and $\boldsymbol{u}^{(0)}$ and the learnable parameters of the NNs (i.e., weights of the linear layers and the Bernstein coefficients $\boldsymbol{c}$ for each neuron), we will update the bounds $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$ with each step of

Algorithm 1: Training step of an L-layer DeepBern-Net $\mathcal{NN}$

Given: Training Batch $(\mathcal{X}, \boldsymbol{t})$ and input bounds $[\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)}]$
Initialize all parameters
Set the learning rate α
▷ Forward propagation
Set $\boldsymbol{y}^{(0)} = \mathcal{X}$
Set $\mathcal{B}^{(0)} = [\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)}]$
for i = 1....L **do**
  if $layer\ i$ is Bernstein activation **then**
    $\boldsymbol{l}^{(i)}, \boldsymbol{u}^{(i)} \leftarrow \mathcal{B}^{(i-1)}$ ▷ Store Input bounds of the Bernstein layer
    for each neuron z in $layer\ i$ **do**
      Let $\boldsymbol{c_z^{(i)}}$ be the Bernstein coefficients for neuron z of the i-th layer
      $\mathcal{B}_z^{(i)} \leftarrow [\min_j c_{zj}^{(i)}, \max_j c_{zj}^{(i)}]$
    end for
    $\mathcal{B}^{(i)} \leftarrow [\mathcal{B}_0^{(i)}, \mathcal{B}_1^{(i)}, ..., \mathcal{B}_m^{(i)}]$ ▷ m denotes the number of neurons in layer i
  else
    $\mathcal{B}^{(i)} \leftarrow$IBP($\mathcal{B}^{(i-1)}$)
  end if
  $\boldsymbol{y}^{(i)} \leftarrow$ forward($\boldsymbol{y}^{(i-1)}$) ▷ Regular forward step
end for
▷ Backpropagation
Compute the loss function: $\mathcal{L}(\boldsymbol{y}^{(L)}, \boldsymbol{t})$
Compute the gradients with respect to all model parameters (including Bernstein coefficients)
for each Parameter θ **do** ▷ Weights, biases, and Bernstein coefficients c_k
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
end for

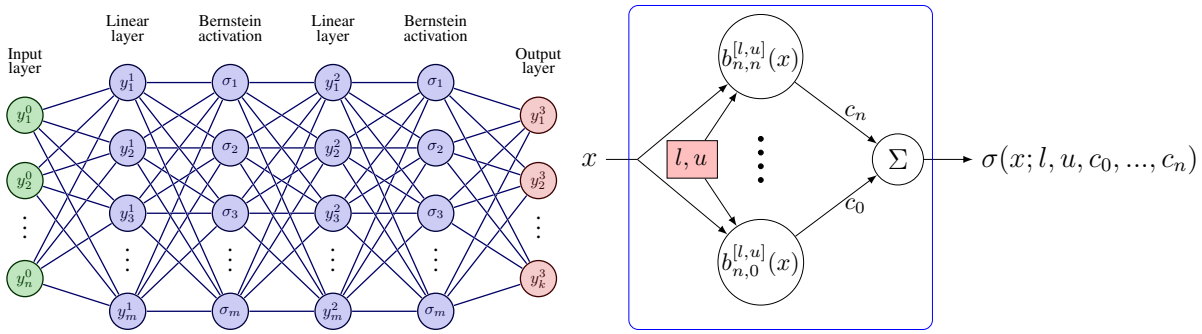

Figure 1: (Left) shows the structure of a DeepBern-Nets with two hidden layers. DeepBern-Nets are similar to Feed Forward NNs except that the activation function is a Bernstein polynomial. (Right) shows a simplified computational graph of a degree n Bernstein activation. The Bernstein basis is evaluated at the input x using l and u computed during training, and the output is then computed as a linear combination of the basis functions weighted by the learnable Bernstein coefficients c_k.

training by propagating $\boldsymbol{l}^{(0)}$ and $\boldsymbol{u}^{(0)}$ through all the layers in the network. Unlike conventional non-linear activation functions where symbolic bound propagation relies on linear relaxation techniques (Wang et al. 2018a,b), the Bernstein polynomial enclosure property allows us to bound the output of an n-th order Bernstein activation in $\mathcal{O}(n)$ operations (Algorithm 1-line 12). We start by reviewing the enclosure property of Bernstein polynomials as follows.

Property 1 (Enclosure of Range (Titi 2019))**.** The enclosure property of Bernstein polynomials states that for a given polynomial $P_n^{[l,u]}(x)$ of degree n in Bernstein form on an interval $[l, u]$, the polynomial lies within the convex hull of its Bernstein coefficients. In other words, the Bernstein polynomial is bounded by the minimum and maximum values of its coefficients c_k regardless of the input x.

$$\min_{0\le k\le n} c_k \le P_n^{[l,u]}(x) \le \max_{0\le k\le n} c_k, \qquad \forall x \in [l, u]. \quad (4)$$

Algorithm 1 outlines how to use the enclosure property to propagate the bounds from one layer to another for a single training step in an L-layer DeepBern-Net. In contrast to normal training, we calculate the worst-case bounds for the inputs to all Bernstein layers by propagating the bounds from the previous layers. Such bound propagation can be done for linear layers using interval arithmetic (Liu et al. 2021)—referred to in Algorithm 1-line 16 as Interval Bound Propagation (IBP)—or using Property 1 for Bernstein layers (Algorithm 1-Line 12). We store the resulting bounds for each Bernstein activation function. Then, we perform the regular forward step. The parameters are then updated using vanilla backpropagation, just like conventional NNs. During inference, we directly use the stored layer-wise bounds $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$ (computed during training) to propagate any input through the network. In Appendix C[1], we show that the overhead of computing the bounds $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$ during training adds between $0.2\times$ to $5\times$ overhead for the training, depending on the order n of the Bernstein activation function and the size of the network.

[1]Extended version at https://arxiv.org/abs/2305.13508

Stable training of DeepBern-Nets. Using polynomials as activation functions in deep NNs has attracted several researchers' attention in recent years (Wang, Chen, and Ng 2022; Gottemukkula 2020). A major drawback of using polynomials of arbitrary order is their unstable behavior during training due to exploding gradients–which is prominent with the increase in order (Gottemukkula 2020). In particular, for a general nth order polynomial in power series $f_n(x) = w_0 + w_1 x + \ldots + w_n x^n$, its derivative is $df_n(x)/dx = w_1 + \ldots + n w_n x^{n-1}$. Hence training a deep NN with multiple polynomial activation functions suffers from exploding gradients as the gradient scales exponentially with the increase in the order n for $x > 1$.

Luckily, and thanks to the unique properties of Bernstein polynomials, DeepBern-Net does not suffer from such a limitation as captured in the next result, whose proof is given in Appendix A.

Proposition 1. Consider the Bernstein activation function $\sigma(x; l, u, \boldsymbol{c})$ of arbitrary order n. The following holds:

1. $\left|\frac{d}{dx}\sigma(x; l, u, \boldsymbol{c})\right| \le 2n \max_{k\in\{0,\ldots,n\}} |c_k|$,
2. $\left|\frac{d}{dc_i}\sigma(x; l, u, \boldsymbol{c})\right| \le 1 \quad$ for all $i \in \{0, \ldots, n\}$.

Proposition 1 ensures that the gradients of the proposed Bernstein-based activation function depend only on the value of the learnable parameters $\boldsymbol{c} = (c_0, \ldots, c_n)$. Hence, the gradients do not explode for $x > 1$. This feature is not enjoyed by the polynomial activation functions in (Gottemukkula 2020) and leads to better stable training properties. Moreover, one can control these gradients by adding a regularizer–to the objective function–that penalizes high values of c_k, which is common for other learnable parameters, i.e., weights of the linear layer. Proof of Proposition 1 is in Appendix A

3 Bern-IBP: Certification Using Bernstein Interval Bound Propagation

Certification of Global Properties Using Bern-IBP

We consider the certification of global properties of NNs. Global properties need to be held true for the entire input

domain $\mathcal{D}$ of the network. For simplicity of presentation, we will assume that the global property we want to prove takes the following form:

$$\forall \boldsymbol{y^{(0)}} \in \mathcal{D} \Longrightarrow y^{(L)} = \mathcal{NN}(\boldsymbol{y}^{(0)}) > 0 \tag{5}$$

where $y^{(L)}$ is a scalar output and $\mathcal{NN}$ is the NN of interest. Examples of such global properties include the stability of NN-controlled systems (Wu, Chen, and Chen 2022) as well as global individual fairness (Khedr and Shoukry 2022).

In this paper, we focus on the incomplete certification of such properties. In particular, we certify properties of the form (5) by checking the lower/upper bounds of the NN. To that end, we define the lower $\mathcal{L}$ and upper $\mathcal{U}$ bounds of the NN within the domain $\mathcal{D}$ as any real numbers that satisfy:

$$\begin{aligned} \mathcal{L}\left(\mathcal{NN}(\boldsymbol{y}^{(0)}), \mathcal{D}\right) &\leq \min_{\boldsymbol{y}^{(0)} \in \mathcal{D}} \mathcal{NN}(\boldsymbol{y}^{(0)}), \\ \mathcal{U}\left(\mathcal{NN}(\boldsymbol{y}^{(0)}), \mathcal{D}\right) &\geq \max_{\boldsymbol{y}^{(0)} \in \mathcal{D}} \mathcal{NN}(\boldsymbol{y}^{(0)}) \end{aligned} \tag{6}$$

Incomplete certification of (5) is equivalent to checking if $\mathcal{L}\left(\mathcal{NN}(\boldsymbol{y}^{(0)}), \mathcal{D}\right) > 0$. Thanks to the Enclosure of Range (Property 1) of DeepBern-Nets, one can check the condition $\mathcal{L}\left(\mathcal{NN}(\boldsymbol{y}^{(0)}), \mathcal{D}\right) > 0$ in constant time, i.e., $\mathcal{O}(1)$, by simply checking the minimum Bernstein coefficients of the output layer.

Certification of Local Properties Using Bern-IBP

Local properties of NNs are the ones that need to be held for subsets S of the input domain $\mathcal{D}$, i.e.,

$$\forall \boldsymbol{y^{(0)}} \in S \subset \mathcal{D} \Longrightarrow y^{(L)} = \mathcal{NN}(\boldsymbol{y}^{(0)}) > 0 \tag{7}$$

Examples of local properties include adversarial robustness and the safety of NN-controlled vehicles (Sun, Khedr, and Shoukry 2019; Kochdumper et al. 2023; Santa Cruz and Shoukry 2022). Similar to global properties, we are interested in incomplete certification by checking whether $\mathcal{L}\left(\mathcal{NN}(\boldsymbol{y}^{(0)}), S\right) > 0$.

The output bounds stored in the Bernstein activation functions are the worst-case bounds for the entire input domain $\mathcal{D}$. However, for certifying local properties over $S \subset \mathcal{D}$, we need to refine these output bounds on the given sub-region S. To that end, for a Bernstein activation layer k with input bounds $[\boldsymbol{l}^{(k)}, \boldsymbol{u}^{(k)}]$ (computed and stored during training), we can obtain tighter output bounds thanks to the following subdivision property of Bernstein polynomials.

Property 2 (Subdivision (Titi 2019))**.** Given a Bernstein polynomial $P_n^{[l,u]}(x)$ of degree n on the interval $[l, u]$, the coefficients of the same polynomial on subintervals $[l, \alpha]$ and $[\alpha, u]$ with $\alpha \in [l, u]$ can be computed as follows. First, compute the intermediate coefficients c_j^k for $k = 0, ..., n$ and $j = k, ..., n$

$$\begin{aligned} c_j^k &= \begin{cases} c_j & \text{if } k = 0 \\ (1-\tau)c_{j-1}^{k-1} + \tau c_j^{k-1} & \text{if } k > 0 \end{cases}, \\ c_i' &= c_i^i, \quad c_i'' = c_n^{n-i} \quad i = 0 \dots n, \end{aligned} \tag{8}$$

where $\tau = \frac{\alpha - l}{u - l}$. Next, the polynomials defined on each of the subintervals $[l, \alpha]$ and $[\alpha, u]$ are:

$$P_n^{[l,\alpha]}(x) = \sum_{k=0}^{n} c_k' b_{n,k}^{[l,\alpha]}(x), \quad P_n^{[\alpha,u]}(x) = \sum_{k=0}^{n} c_k'' b_{n,k}^{[\alpha,u]}(x).$$

Indeed, we can apply the Subdivision property twice to compute the coefficients of the polynomial $P_n^{[\alpha,\beta]}$. Computing the coefficients on the subintervals allows us to tightly bound the polynomial using property 1. Therefore, given a DeepBern-Net trained on $\mathcal{D} = [\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)}]$, we can compute tighter bounds on the subregion $S = [\hat{\boldsymbol{l}}^{(0)}, \hat{\boldsymbol{u}}^{(0)}]$ by applying the subdivision property (Property 2) to compute the Bernstein coefficients on the sub-region S, and then use the enclosure property (Property 1) to compute tight bounds on the output of the activation equivalent to the minimum and maximum of the computed Bernstein coefficients. We do this on a layer-by-layer basis until we reach the output of the NN. Implementation details of this approach is given in Appendix B.

4 Experiments

Implementation: Our framework has been developed in Python, and is designed to facilitate the training of DeepBern-Nets and certify local properties such as Adversarial Robustness and certified training. We use PyTorch (Paszke et al. 2019) for all neural network training tasks. To conduct our experiments, we utilized a single GeForce RTX 2080 Ti GPU in conjunction with a 24-core Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. Only 8 cores were utilized for our experiments.

Experiment 1: Certification of Adversarial Robustness

The first experiment assesses the tightness of bounds on the NN output and its implications for certifying NN properties. To that end, we use the application of adversarial robustness, where we aim to certify that a NN model is not susceptible to adversarial examples within a defined perturbation set. The results in (Li, Xie, and Li 2020; Müller et al. 2022a) show that state-of-the-art IBP algorithms fail to certify the robustness of NNs trained with Projected Gradient Descent (PGD), albeit being robust, due to the excessive errors in the computed bounds, which forces designers to use computationally expensive sound and complete algorithms. Thanks to the properties of DeepBern-Nets, the bounds computed by Bern-IBP are tight enough to certify the robustness of NNs without using computationally expensive sound and complete tools. To that end, we trained several NNs using the MNIST (LeCun 1998) and CIFAR-10 (Krizhevsky and Hinton 2014) datasets using PGD. We trained both Fully Connected Neural Networks (FCNN) and Convolutional Neural Networks (CNNs) on these datasets with Bernstein polynomials of orders $2, 3, 4, 5$, and 6. For detailed information regarding the model architectures, please refer to Appendix C. Further information about the training procedure can be found in Appendix C.

Formalizing Adversarial Robustness as a Local Property Given a NN model $\mathcal{NN} : [0,1]^d \rightarrow \mathbb{R}^o$, a concrete input $\boldsymbol{x_n}$, a target class t, and a perturbation parameter ϵ, the adversarial robustness problem asks that the NN output be the target class t for all the inputs in the set $\{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x_n}\|_\infty \leq \epsilon\}$. In other words, a NN is robust whenever:

$$\forall \boldsymbol{x} \in S(\boldsymbol{x_n}, \epsilon) = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x_n}\|_\infty \leq \epsilon\} \implies \mathcal{NN}(\boldsymbol{x})_t > \mathcal{NN}(\boldsymbol{x})_i,\ i \neq t$$

where $\mathcal{NN}(\boldsymbol{x})_t$ is the NN output for the target class and $\mathcal{NN}(\boldsymbol{x})_i$ is the NN output for any class i other that t. To certify the robustness of a NN, one can compute a lower bound on the adversarial robustness $\mathbb{L}_{\text{robust}}$ for all classes $i \neq t$ as:

$$\begin{aligned}\mathbb{L}_{\text{robust}}(\boldsymbol{x_n}, \epsilon) &= \min_{i \neq t} \Big(\mathcal{L}\big(\mathcal{NN}(\boldsymbol{x})_t, S(\boldsymbol{x_n}, \epsilon)\big) \\ &\qquad - \mathcal{U}\big(\mathcal{NN}(\boldsymbol{x})_i, S(\boldsymbol{x_n}, \epsilon)\big) \Big) \\ &\leq \min_{i \neq t} \Big(\min_{\boldsymbol{x} \in S(\boldsymbol{x_n}, \epsilon)} \mathcal{NN}(\boldsymbol{x})_t - \mathcal{NN}(\boldsymbol{x})_i \Big)\end{aligned} \tag{9}$$

Indeed, the NN is robust whenever $\mathbb{L}_{\text{robust}} > 0$. Nevertheless, the tightness of the bounds $\mathcal{L}(\mathcal{NN}(\boldsymbol{x})_t, S(\boldsymbol{x_n}, \epsilon))$ and $\mathcal{U}(\mathcal{NN}(\boldsymbol{x})_i, S(\boldsymbol{x_n}, \epsilon))$ plays a significant role in the ability to certify the NN robustness. The tighter these bounds, the higher the ability to certify the NN robustness.

Experiment 1.1: Tightness of Output Bounds – Bern-IBP vs IBP For each trained neural network, we compute the lower bound on robustness $\mathbb{L}_{\text{robust}}(\boldsymbol{x_n}, \epsilon)$ using Bern-IBP and using state-of-the-art Interval Bound Propagation (IBP) that does not take into account the properties of DeepBern-Nets. In particular, for this experiment, we used auto_LiRPA(Wang et al. 2021), a tool that is part of $\alpha\beta$-CROWN(Wang et al. 2021)—the winner of the 2022 Verification of Neural Network (VNN) competition (Müller et al. 2022a). Figure 2 shows the difference between the bound $\mathbb{L}_{\text{robust}}(\boldsymbol{x_n}, \epsilon)$ computed by Bern-IBP and the one computed by IBP using a semi-log scale. The raw data for the adversarial robustness bound $\mathbb{L}_{\text{robust}}(\boldsymbol{x_n}, \epsilon)$ for both Bern-IBP and IBP is given in Appendix C.

The results presented in Figure 2 clearly demonstrate that Bern-IBP yields significantly tighter bounds in comparison to IBP. Figure 2 also shows that for all values of ϵ, the bounds computed using IBP become exponentially looser as the order of the Bernstein activations increase, unlike the bounds computed with Bern-IBP, which remain precise even for higher-order Bernstein activations or larger values of ϵ. The raw data in Appendix C provide a clearer view on the superiority of computing $\mathbb{L}_{\text{robust}}(\boldsymbol{x_n}, \epsilon)$ using Bern-IBP compared to IBP.

Experiment 1.2: Certification of Adversarial Robustness Using Bern-IBP Next, we show that the superior precision of bounds calculated using Bern-IBP can lead to efficient certification of adversarial robustness. Here, we define the certified accuracy of the NN as the percentage of the data points (in the test dataset) for which an adversarial input can not change the class (the output of the NN). Table 1 contrasts the certified accuracy for the adversarially-trained (using 100-step PGD) DeepBern-Nets of orders 2, 4, and 6, using both IBP and Bern-IBP methods and varying values of ϵ. As observed by the table, IBP fails to certify the robustness of all the NNs. On the other hand, Bern-IBP achieved high certified accuracy for all the NNs with varying values of ϵ. Finally, we use the methodology reported in (Wang et al. 2021) to upper bound the certified accuracy using 100-step PGD attack.

It is essential to mention that IBP's inability to certify the robustness of NNs is not unique to DeepBern-Nets. In particular, as shown in (Li, Xie, and Li 2020; Müller et al. 2022a), most certifiers struggle to certify the robustness of ReLU NNs when trained with PGD. This suggests the power of DeepBern-Nets, which can be efficiently certified—in a few seconds even for NNs with millions of parameters, as shown in Table 1—using incomplete certifiers thanks to the ability of Bern-IBP to compute tight bounds.

Experiment 2: Certified Training Using Bern-IBP

In this experiment, we demonstrate that the tight bounds calculated by Bern-IBP can be utilized for certified training, achieving comparable results to SOTA methods when trained in a controlled training setup as explained in the SOK (Li, Xie, and Li 2020) benchmark. Importantly, we don't claim achieving the state-of-the-art best results on these datasets which usually requires very long training with sophisticated warmup and ϵ schedules. Instead, we follow the training setup used in SOK (fixed training epochs, and no ϵ-schedule) to demonstrate the effecivness of Bern-IBP's tight bounds on certified training. We trained neural networks with the same architectures as those in the benchmark to maintain a similar (not exact) number of parameters, with the polynomial order serving as an additional hyperparameter. The training objective adheres to the certified training literature (Zhang et al. 2019), incorporating the bound on the robustness loss in the objective as follows:

$$\begin{aligned}\min_{\theta} \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\in(X,Y)} \Big[&(1-\lambda)\mathcal{L}_{\text{CE}}(\mathcal{NN}_\theta(\boldsymbol{x}), y; \theta) \\ &+ \lambda \mathcal{L}_{\text{RCE}}(S(\boldsymbol{x}, \epsilon), y; \theta)) \Big],\end{aligned} \tag{10}$$

where $\boldsymbol{x}$ is a data point, y is the ground truth label, $\lambda \in [0,1]$ is a weight to control the certified training regularization, $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss, θ is the NN parameters, and $\mathcal{L}_{\text{RCE}}$ is computed by evaluating $\mathcal{L}_{\text{CE}}$ on the upper bound of the logit differences computed(Zhang et al. 2019) using a bounding method.

For DeepBern-Nets, $\mathcal{L}_{\text{RCE}}$ is computed using Bern-IBP during training, while the networks in the SOK benchmark are trained using CROWN-IBP (Zhang et al. 2019). Table 2 illustrates that employing Bern-IBP bounds for certified training yields state-of-the-art certified accuracy (certified with Bern-IBP) on these datasets, comparable to—or in many cases surpassing—the performance of ReLU networks. The primary advantage of using Bern-IBP lies in its

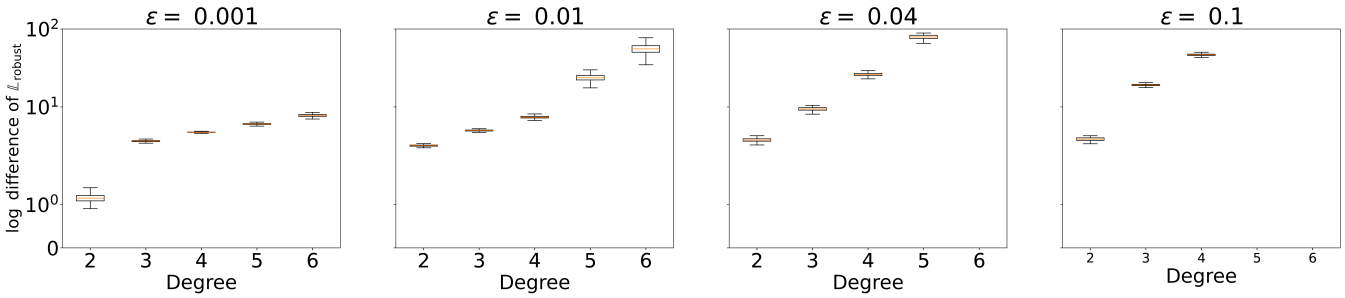

Figure 2: A visual representation of the tightness of bounds computed using Bern-IBP compared to IBP. The figure shows the log difference between $\mathbb{L}_{\text{robust}}$ computed using Bern-IBP and IBP for NNs with varying orders of and different values of ϵ. The figure demonstrates the enhanced precision and scalability of the Bern-IBP method in computing tighter bounds, even for higher-order Bernstein activations and larger values of ϵ, as compared to the naive IBP method.

Dataset	Model (# of params)	Test acc. (%)	ϵ	IBP		Bern-IBP		U.B (PGD)
				Time (s)	Certified acc. (%)	Time (s)	Certified acc. (%)	Certified acc. (%)
MNIST	CNNa_4 (190,426)	97.229	0.01	3.45	0	1.43	88.69	95.97
			0.03	3.41	0	1.42	72.12	92.53
			0.1	3.26	0	1.39	65.22	75.27
	CNNb_2 (905,882)	97.14	0.01	4.38	0	2.07	80.21	95.42
			0.03	4.58	0	2.11	56.49	90.57
			0.1	4.61	0	1.97	72.35	78.6
CIFAR-10	CNNa_6 (258,626)	46.77	1/255	3.29	0	1.82	27.74	33.53
			2/255	3.25	0	1.83	33.49	35.81
	CNNb_4 (1,235,994)	54.66	1/255	5.17	0	4.45	28.55	42.86
			2/255	5.14	0	4.33	14.7	36.73

Table 1: A comparison of certified accuracy and verification time for neural networks with Bernstein polynomial activations using both IBP and Bern-IBP methods and varying values of ϵ. The table also presents the upper bound on certified accuracy calculated using a 100-step PGD attack. The results highlight the superior performance of Bern-IBP in certifying robustness properties compared to IBP.

ability to compute highly precise bounds using a computationally cheap method, unlike the more sophisticated bounding methods for ReLU networks, such as α-Crown. For more details about the exact architecture of the NNs, please refer to Appendix C

Experiment 3: Tight Reachability Analysis of NN-Controlled Quadrotor Using Bern-IBP

In this experiment, we study the application-level impact of using Bernstein polynomial activations in comparison to ReLU activations with respect to the tightness of reachable sets in the context of safety-critical applications. Specifically, we consider a 6D linear dynamics system $\dot{x} = Ax + Bu$ representing a Quadrotor (used in (Everett et al. 2021; Hu et al. 2020; Lopez et al. 2019)), controlled by a nonlinear NN controller where $u = \mathcal{NN}(x)$. To ensure a fair comparison, both sets of networks are trained on the same datasets, using the same architectures and training procedures. The only difference between the two sets of networks is the activation function used (ReLU vs. Bernstein polynomial).

After training, we perform reachability analysis with horizon $T = 6$ on each network using the respective bounding methods: Crown and α-Crown for ReLU networks and the proposed Bern-IBP for Bernstein polynomial networks. We compute the volume of the reachable sets after each step for each network. The results are visualized in Figure 3, comparing the error in the volume of the reachable sets for both ReLU and Bernstein polynomial networks. The error is computed with respect to the true volume of the reachable set for each network, which is computed by heavy sampling. As shown in Figure 3, using Bern-IBP on the NN with Bernstein polynomial can lead to much tighter reachable sets compared to SOTA bounding methods for ReLU networks. This experiment provides insights into the potential benefits of using Bernstein polynomial activations for improving the tightness of reachability bounds, which can have significant implications for neural network certification for safety-critical systems.

5 Related Work

Neural Network verification. NN verification is an active field of research that focuses on developing techniques to verify the correctness and robustness of neural networks. Various methods have been proposed for NN verification to provide rigorous guarantees on the behavior of NNs and detect potential vulnerabilities such as adversarial examples

Model	MNIST Certified acc. (%)				CIFAR-10 Certified acc. (%)			
	$\epsilon = 0.1$		$\epsilon = 0.3$		$\epsilon = 2/255$		$\epsilon = 8/255$	
	DeepBern-Net (%)	SOK (%)	DeepBern-Net (%)	SOK (%)	DeepBern-Net (%)	SOK (%)	DeepBern-Net (%)	SOK (%)
FCNNa	**72**	68	**31**	25	**38**	33	**28**	27
FCNNb	**86**	85	**57**	54	**39**	37	**26**	25
FCNNc	**80**	80	**51**	22	**36**	32	**31**	30
CNNa	**95**	95	82	**88**	45	**46**	31	**34**
CNNb	**95**	94	77	**85**	**49**	49	**37**	35
CNNc	87	**89**	72	**87**	38	**51**	32	**38**

Table 2: A comparison of certified accuracy for NNs with Bernstein polynomial activations versus ReLU NNs as in the SOK benchmark (Li, Xie, and Li 2020). The certified accuracy is computed using Bern-IBP for NNs with polynomial activations, and the method yielding highest certified accuracy as reported in SOK for ReLU NNs. The table highlights the effectiveness of Bern-IBP in achieving competitive certification while utilizing a very computationally cheap method for tight bound computation.

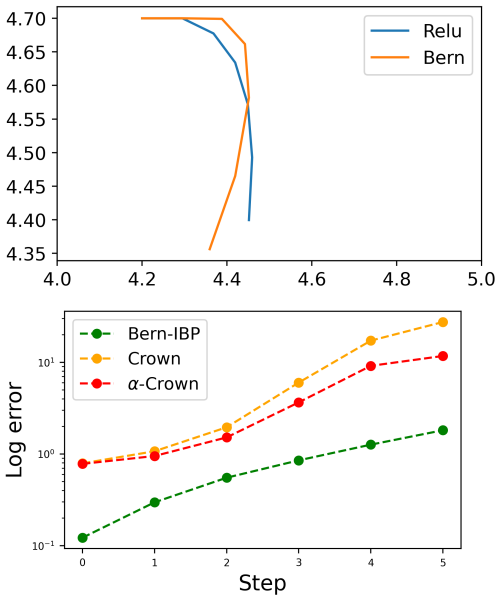

Figure 3: (Top) The trajectory of the Quadrotor for the ReLU and Bernstein polynomial networks. (Bottom) the error in the reachable set volume $e = (\hat{V} - V)/V$ for each of the networks after each step. $\hat{V}$ is the estimated volume using the respective bounding method and V is the true volume of the reachable set using heavy sampling

and unfairness. These methods use techniques such as abstract interpretation (Ferrari et al. 2022), Satisfiability Modulo Theory (SMT) (Katz et al. 2019), Reachability Analysis (Bak 2021; Tran et al. 2020) and Mixed-Integer Linear Programming (MILP) (Lomuscio and Maganti 2017; Tjeng, Xiao, and Tedrake 2019; Bunel et al. 2020; Anderson et al. 2020). Many tools also rely on optimization and linear relaxation techniques (Wang et al. 2021; Khedr, Ferlez, and Shoukry 2021; Henriksen and Lomuscio 2021) to speedup the verification. Another line of work (Wan et al. 2023; Fatnassi et al. 2023) uses higher order relaxation such as Bernstein Polynomials to certify NNs. However, frameworks for NN verification often result in loose bounds during the relaxation process or are computationally expensive, particularly for large-scale networks.

Polynomial activations. NNs with polynomial activations have been studied in (Gottemukkula 2020). Theoretical work was established on their expressiveness (Kileel, Trager, and Bruna 2019) and their universal approximation property (Kidger and Lyons 2020) is established under certain conditions. However, to the best of our knowledge, using Bernstein polynomials in Deep NNs and their impact on NN certification has not been explored yet.

Polynomial Neural Networks. A recent work(Chrysos et al. 2021) proposed a new class of approximators called Π-nets, which is based on polynomial expansion. Empirical evidence has shown that Π-nets are highly expressive and capable of producing state-of-the-art results in a variety of tasks, including image, graph, and audio processing, even without the use of non-linear activation functions. When combined with activation functions, they have been demonstrated to achieve state-of-the-art performance in challenging tasks such as image generation, face verification, and 3D mesh representation learning. A framework for certifying such networks using α-convexification was introduced in (Abad Rocamora et al. 2022).

Constrained Neural Networks. 1-Lipschitz NNs constrain the Lipschitz constant of the NNs, thus allowing some guarantees on their behaviour (e.g. robustness). We think it would be interesting to study how DeepBern-Nets relate to these types of NNs. We leave this to future work.

Acknowledgments

This work was partially sponsored by the NSF Awards #2002405, #2313104, and #2139781 and the C3.AI Digital Transformation Institute.

References

Abad Rocamora, E.; Sahin, M. F.; Liu, F.; Chrysos, G.; and Cevher, V. 2022. Sound and complete verification of polynomial networks. *Advances in Neural Information Processing Systems*, 35: 3517–3529.

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1): 3–39.

Bak, S. 2021. Nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In *NASA Formal Methods: 13th International Symposium, NFM 2021, Virtual Event, May 24–28, 2021, Proceedings*, 19–36. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-76383-1.

Bunel, R.; Lu, J.; Turkaslan, I.; Kohli, P.; Torr, P.; and Mudigonda, P. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42): 1–39.

Chrysos, G. G.; Moschoglou, S.; Bouritsas, G.; Deng, J.; Panagakis, Y.; and Zafeiriou, S. 2021. Deep polynomial neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 44(8): 4021–4034.

Everett, M.; Habibi, G.; Sun, C.; and How, J. P. 2021. Reachability analysis of neural feedback loops. *IEEE Access*, 9: 163938–163953.

Farouki, R.; and Rajan, V. 1987. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3): 191–216.

Farouki, R. T. 2012. The Bernstein Polynomial Basis: A Centennial Retrospective. *Comput. Aided Geom. Des.*, 29(6): 379–419.

Fatnassi, W.; Khedr, H.; Yamamoto, V.; and Shoukry, Y. 2023. BERN-NN: Tight Bound Propagation For Neural Networks Using Bernstein Polynomial Interval Arithmetic. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, 1–11.

Ferrari, C.; Muller, M. N.; Jovanovic, N.; and Vechev, M. 2022. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Gottemukkula, V. 2020. Polynomial activation functions. https://openreview.net/pdf?id=rkxsgkHKvH.

Henriksen, P.; and Lomuscio, A. 2021. DEEPSPLIT: An Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis. In *IJCAI*, 2549–2555.

Hu, H.; Fazlyab, M.; Morari, M.; and Pappas, G. J. 2020. Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In *2020 59th IEEE conference on decision and control (CDC)*, 5929–5934. IEEE.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Majumdar, R.; and Kunčak, V., eds., *Computer Aided Verification*, Lecture Notes in Computer Science, 97–117. Springer International Publishing. ISBN 978-3-319-63387-9.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljić, A.; et al. 2019. The marabou framework for verification and analysis of deep neural networks. In Dillig, I.; and Tasiran, S., eds., *Computer Aided Verification*, 443–452. Springer International Publishing.

Khedr, H.; Ferlez, J.; and Shoukry, Y. 2021. Peregrinn: Penalized-relaxation greedy neural network verifier. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*, 287–300. Springer.

Khedr, H.; and Shoukry, Y. 2022. Certifair: A framework for certified global fairness of neural networks. *arXiv preprint arXiv:2205.09927*.

Kidger, P.; and Lyons, T. 2020. Universal approximation with deep narrow networks. In *Conference on learning theory*, 2306–2327. PMLR.

Kileel, J.; Trager, M.; and Bruna, J. 2019. On the expressive power of deep polynomial neural networks. *Advances in neural information processing systems*, 32.

Kochdumper, N.; Krasowski, H.; Wang, X.; Bak, S.; and Althoff, M. 2023. Provably safe reinforcement learning via action projection using reachability analysis and polynomial zonotopes. *IEEE Open Journal of Control Systems*, 2: 79–92.

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

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

LeCun, Y. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

Li, L.; Xie, T.; and Li, B. 2020. Sok: Certified robustness for deep neural networks. *arXiv preprint arXiv:2009.04131*.

Liu, C.; Arnon, T.; Lazarus, C.; Strong, C.; Barrett, C.; Kochenderfer, M. J.; et al. 2021. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4): 244–404.

Lomuscio, A.; and Maganti, L. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*.

Lopez, D. M.; Musau, P.; Tran, H.-D.; and Johnson, T. T. 2019. Verification of Closed-loop Systems with Neural Network Controllers. In Frehse, G.; and Althoff, M., eds., *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, 201–210. EasyChair.

Lyu, Z.; Guo, M.; Wu, T.; Xu, G.; Zhang, K.; and Lin, D. 2021. Towards evaluating and training verifiably robust neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4308–4317.

Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A. 2021. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6): 1–35.

Müller, M. N.; Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2022a. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *arXiv preprint arXiv:2212.10376*.

Müller, M. N.; Eckert, F.; Fischer, M.; and Vechev, M. 2022b. Certified Training: Small Boxes are All You Need. *arXiv preprint arXiv:2210.04871*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Qian, W.; Riedel, M. D.; and Rosenberg, I. 2011. Uniform approximation and Bernstein polynomials with coefficients in the unit interval. *European Journal of Combinatorics*, 32(3): 448–463.

Santa Cruz, U.; and Shoukry, Y. 2022. NNLander-VeriF: A neural network formal verification framework for vision-based autonomous aircraft landing. In *NASA Formal Methods: 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24–27, 2022, Proceedings*, 213–230. Springer.

Song, D.; Eykholt, K.; Evtimov, I.; Fernandes, E.; Li, B.; Rahmati, A.; Tramer, F.; Prakash, A.; and Kohno, T. 2018. Physical adversarial examples for object detectors. In *Proceedings of the 12th USENIX Conference on Offensive Technologies*, WOOT'18. USENIX Association.

Sun, X.; Khedr, H.; and Shoukry, Y. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 147–156.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Titi, J. 2019. *Matrix Methods for the Tensorial and Simplicial Bernstein Forms with Application to Global Optimization*. Ph.D. thesis, Universität Konstanz.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. arXiv:1711.07356.

Tran, H.-D.; Yang, X.; Manzanas Lopez, D.; Musau, P.; Nguyen, L. V.; Xiang, W.; Bak, S.; and Johnson, T. T. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In Lahiri, S. K.; and Wang, C., eds., *Computer Aided Verification*, 3–17. Springer International Publishing.

Wan, Y.; Zhou, W.; Fan, J.; Wang, Z.; Li, J.; Chen, X.; Huang, C.; Li, W.; and Zhu, Q. 2023. POLAR-Express: Efficient and Precise Formal Reachability Analysis of Neural-Network Controlled Systems. *arXiv preprint arXiv:2304.01218*.

Wang, J.; Chen, L.; and Ng, C. W. W. 2022. A New Class of Polynomial Activation Functions of Deep Learning for Precipitation Forecasting. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, 1025–1035. New York, NY, USA: Association for Computing Machinery. ISBN 9781450391320.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient formal safety analysis of neural networks. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31, 6367–6377.

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

Wang, S.; Zhang, H.; Xu, K.; Lin, X.; Jana, S.; Hsieh, C.-J.; and Kolter, J. Z. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34: 29909–29921.

Wu, W.; Chen, J.; and Chen, J. 2022. Stability Analysis of Systems with Recurrent Neural Network Controllers. *IFAC-PapersOnLine*, 55(12): 170–175. 14th IFAC Workshop on Adaptive and Learning Control Systems ALCOS 2022.

Xu, D.; Yuan, S.; Zhang, L.; and Wu, X. 2018. Fairgan: Fairness-aware generative adversarial networks. In *2018 IEEE International Conference on Big Data (Big Data)*, 570–575. IEEE.

Yang, Y.; and Rinard, M. 2019. Correctness verification of neural networks. *arXiv preprint arXiv:1906.01030*.

Zhang, B.; Jiang, D.; He, D.; and Wang, L. 2022. Rethinking Lipschitz Neural Networks for Certified L-infinity Robustness. *arXiv preprint arXiv:2210.01787*.

Zhang, B. H.; Lemoine, B.; and Mitchell, M. 2018. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 335–340.

Zhang, H.; Chen, H.; Xiao, C.; Gowal, S.; Stanforth, R.; Li, B.; Boning, D.; and Hsieh, C.-J. 2019. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*.