Dynamic Back-Substitution in Bound-Propagation-Based Neural Network Verification

Panagiotis Kouvaros[1,2], **Benedikt Brückner**[1, 3], **Patrick Henriksen**[1], **Alessio Lomuscio**[1,3]
[1]Safe Intelligence, UK
[2]Department of Information Technologies, University of Limassol, Cyprus
[3]Department of Computing, Imperial College London, UK
pkouvaros@uol.ac.cy, benedikt@safeintelligence.ai, patrick@safeintelligence.ai, alessio@safeintelligence.ai

Abstract

We improve the efficacy of bound-propagation-based neural network verification by reducing the computational effort required by state-of-the-art propagation methods without incurring any loss in precision. We propose a method that infers the stability of ReLU nodes at every step of the back-substitution process, thereby dynamically simplifying the coefficient matrix of the symbolic bounding equations. We develop a heuristic for the effective application of the method and discuss its evaluation on common benchmarks where we show significant improvements in bound propagation times.

1 Introduction

The use of neural systems in safety-critical applications such as autonomous systems or aviation requires the ability to provide assurance guarantees on the overall system behaviour. This is known to be problematic for neural networks due to possible fragilities (Szegedy et al. 2014) in distribution and potentially unpredictable behaviour out of distribution. While testing and simulation can aid the validation process for these systems, the approaches fall short of providing the deterministic guarantees that are typically required for certification and deployment.

Formal verification methods for neural networks assess whether different properties hold for a given model. One such property is *local robustness* capturing whether the model is stable in the dense neighbourhood of a given input under a perturbation of interest. For computer vision applications noteworthy perturbations include noise, geometric variations, and photometric perturbations (such as contrast, luminosity and bias-field changes), which are particularly usefull in the modelling of photometric phenomena in applications such as autonomous aviation (Kouvaros et al. 2021). While rapid progress has been made in the area over the past few years, performance remains a key objective to address ever larger models used in applications.

Symbolic interval propagation (Wang et al. 2018b,a; Singh et al. 2018a, 2019b; Zhang et al. 2018) (SIP) has emerged as one of the key approaches for assessing local robustness at scale, both as a subroutine and as the core component of verification algorithms. Over the past 12—24 months, progress on SIP efficiency has comparably slowed down. Yet, it remains of fundamental importance for methods such as SIP to become more efficient. This is particularly the case for large, deep and convolutional models that are heavily used in advanced computer vision applications, including object detection.

The objective of the work reported here is to accelerate SIP. We specifically develop a dynamic back-substitution method for running SIP on networks with ReLU activation functions, which is expected to have a compounding effect on performance when integrated with existing neural network verifiers. It works by reducing the size of the back-substitution equations in SIP at a very low cost and with no precision loss, thereby resulting in faster computation through the model. This is achieved by combining intermediate results from the more expensive back-substitution-based SIP method with symbolic bounds for intermediate layers obtained from the cheaper forward SIP method (which is typically ran to obtain coarse bounds before employing more sophisticated methods). The combination of these intermediate results aims at the identification of stable ReLU neurons, which can be dropped from the list of nodes for which we continue running the expensive SIP. This results in a cost reduction with no precision loss. Since identifying these stable nodes comes at a cost, we also present a heuristic for deciding when to apply our method. Specifically, we make the following contributions.

1. We present the mathematical formulation of DBS-SIP together with a heuristic governing its use.
2. We evaluate the method on a number of computer vision models ranging from 92K to 21.8M parameters.

The results obtained show that while the method provides little to no gains on small models, in the case of large models such as ResNet18 and ResNet34 the method leads to up to a 40% gain compared to the present state-of-the-art. This leads to a considerable increase in the number of queries that can be resolved via SIP through the acceleration of bounding in Branch-and-Bound (BaB) schemes.

Related Work. Algorithms for Neural Network Verification can be divided into complete and incomplete approaches. Complete verifiers are guaranteed to provide a definite answer to the verification problem, but are computationally expensive and may not to scale to large networks.

Incomplete methods are based on relaxations of the verification problem to achieve better efficacy. In doing so, they sacrifice completeness, may produce spurious counterexamples and possibly fail to verify properties for large networks because of exploding overapproximation errors.

Complete methods are usually based on SMT (Ehlers 2017; Katz et al. 2017, 2019) or MILP. While standard MILP solvers were used in early approaches (Lomuscio and Maganti 2017; Cheng, Nührenberg, and Ruess 2017; Kouvaros and Lomuscio 2018; Tjeng, Xiao, and Tedrake 2019; Anderson et al. 2020), speedups were later achieved through customised branching strategies (Bunel et al. 2018; Botoeva et al. 2020; Kouvaros and Lomuscio 2021; Singh et al. 2019c). Incomplete methods make use of Semidefinite Programming (SDP) (Raghunathan, Steinhardt, and Liang 2018; Fazlyab, Morari, and Pappas 2020; Batten et al. 2021; Lan, Zheng, and Lomuscio 2022, 2023; Lan, Brueckner, and Lomuscio 2023). Interval Bound Propagation (IBP) (Gowal et al. 2019) or Symbolic Interval Propagation (SIP) (Wang et al. 2018a,b; Singh et al. 2018b) can often be rendered complete by combining them with a branching strategy.

Symbolic Interval Propagation has been thoroughly explored and is extensively used in the context of the formal verification of neural networks. The method propagates symbolic bounding equations from the input layer of the network to the output layer. A number of improvements of the original SIP algorithm have been introduced. Back-substitution, for example, performs propagation in a reverse manner from the last to the first layer of the network which yields tighter approximations (Zhang et al. 2018; Singh et al. 2019b). The present work improves the back-substitution algorithm without incurring any precision loss by dynamically inferring the stability of ReLU nodes. This can significantly reduce the computational cost by reducing the number of nodes for which back-substitution needs to be executed.

The tightness of the ReLU relaxations required for the SIP process can be improved by considering multiple neurons simultaneously (Singh et al. 2019a) or by explicitly optimising the slopes of the employed bounding functions. Some approaches realise this optimisation via gradient descent (Xu et al. 2021) and others through the utilisation of MILP solvers (Hashemi, Kouvaros, and Lomuscio 2021). Whereas these methods aim at tightening the bounds for the output of the network, the method here proposed achieves a faster computation of the same bounds.

Back-substitution forms an integral part of many complete verifiers. These combine SIP with BaB schemes and put forward a number of heuristics for the acceleration of verification through the improvement of the branching decisions that are made (Bunel et al. 2018; Palma et al. 2021; Henriksen and Lomuscio 2021). The verifiers employ further optimisations, including the Lagrangian relaxations of ReLU split constraints generated during BaB (Wang et al. 2021), and the integration of cuts generated by MILP solvers into the bound propagation process (Zhang et al. 2022). Complete verifiers can benefit from our method by accelerating the back-substitution subroutines.

Note that even verification methods that are not mainly based on SIP, often still utilise it as a subroutine. For example, SMT- and MILP-based methods frequently use SIP to calculate the bounds necessary for ReLU encodings or for other pre-processing purposes (Katz et al. 2019; Kouvaros and Lomuscio 2021). SIP is also employed in a variety of methods for the verification of geometric properties (Balunovic et al. 2019; Batten et al. 2024), and for the abstraction-based robust training of neural networks (Mirman, Gehr, and Vechev 2018; Henriksen and Lomuscio 2023). For any of these methods which employ back-substitution, our novel method can lead to a substantial speed-up in the pre-processing step.

2 Background

In this section we recall neural networks, neural network verification, and symbolic interval propagation.

Feed-forward neural networks. A *feed-forward neural network* (FFNN) is a vector-valued function $f(x_0)\colon \mathbb{R}^{s_0} \to \mathbb{R}^{s_L}$ that composes a sequence of $L \geq 1$ layers $f_0 : \mathbb{R}^{s_0} \to \mathbb{R}^{s_0}$, $f_1 : \mathbb{R}^{s_0} \to \mathbb{R}^{s_1}, \ldots, f_L : \mathbb{R}^{s_{L-1}} \to \mathbb{R}^{s_L}$. We denote by x_i the output of each layer f_i. Every element of x_i is often called a *neuron*. Layer f_0 implements the identity function, i.e., $f_0(x_0) = x_0$, and it is said to be the *input layer*. A layer f_i with $1 \leq i \leq L$ is said to be a *hidden layer* and f_L is said to be the *output layer*. Every (non-input) layer implements for input x_{i-1} either *(i)* an affine transformation $f_i(x_{i-1}) = W_i x_{i-1} + b_i$, for a weight matrix $W_i \in \mathbb{R}^{s_i \times s_{i-1}}$ and a bias vector $b_i \in \mathbb{R}^{s_i}$, or *(ii)* a ReLU activation function $f_i(x_{i-1}) = ReLU(x_{i-1}) = \max(x_{i-1}, 0)$, where the maximum function is applied element-wise. Note that for ease of presentation we separate affine transformations from the ReLU activation function and consider each as a different layer, as opposed to their standard treatment whereby their composition defines a layer. We hereafter we consider classification networks, which classify a given input as belonging to a certain class $c \in \{1, \ldots, s_L\}$, which is given by $c = \arg\max(f(x_0))$.

Verification problem. Given a FFNN, the verification problem is to answer whether its output falls within a linearly definable set (i.e., a set that can be described using a finite set of affine constraints) of outputs for every input within a linearly definable set of inputs.

Definition 1. *Verification problem. Given a FFNN f, a linearly definable set of inputs $\mathcal{X} \subset \mathbb{R}^{s_0}$ and a linearly definable set of outputs $\mathcal{Y} \subset \mathbb{R}^{s_L}$, the verification problem concerns establishing whether $\forall x_0 \in \mathcal{X}\colon f(x_0) \in \mathcal{Y}$.*

We write $(f, \mathcal{X}, \mathcal{Y})$ to denote a verification problem. A widely studied instantiation of the problem is the adversarial robustness problem, whereby the robustness of neural networks to adversarial examples is assessed (Huang et al. 2017; Anderson et al. 2019; Dvijotham et al. 2018; Bastani et al. 2016; Wang et al. 2018a; Katz et al. 2017). Adversarial examples are semantically equivalent inputs that, while differing in only imperceptible perturbations, they are classified differently by a network in question. For instance, the robustness of neural networks with respect to an input x_0 and white noise perturbations can be checked by setting $\mathcal{X} = \{x_0' \mid x_0 - \epsilon \leq x_0' \leq x_0 + \epsilon\}$ and $\mathcal{Y} =$

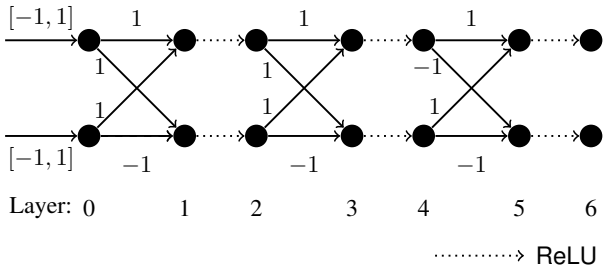


Figure 1: A neural network with six layers. The input neurons are constrained within the range $[-1, 1]$. The weights on the edges represent the weight matrices for the affine layers (layers 1 and 3) which are assumed to have zero bias.

$\{y \mid y = f(x_0'), x_0' \in \mathcal{X}, \forall i \neq c\colon y(i) < y(c)\}$, where c is the class of x_0 and $\epsilon > 0$ is the perturbation radius. Adversarial robustness for photometric transformations, such as brightness, contrast and bias fields (Kouvaros et al. 2023; Henriksen et al. 2021), can be defined in terms of white noise perturbations by prepending to the network in question representations of the transformations as affine layers. We thus hereafter focus our exposition on the adversarial robustness problem with respect to white noise perturbations.

Symbolic Interval Propagation (SIP). SIP is a popular incomplete framework for solving the adversarial robustness problem. SIP methods typically form the backbone of complete verification frameworks, whether these are based on MILP (Kouvaros and Lomuscio 2021; Singh et al. 2019c) or branch-and-bound schemes (Palma et al. 2021; Wang et al. 2021; Henriksen and Lomuscio 2021). The methods propagate linearly definable bounds through the network on a layer-by-layer basis, thereby computing bounds for the output of each of the layers. As detailed below, the bounds for the output layer can often be used to solve the verification problem. The methods include Forward SIP (FSIP) (Wang et al. 2018a) and back-substitution-based SIP (BS-SIP) (Singh et al. 2019b; Zhang et al. 2018). BS-SIP derives tighter bounds than FSIP but at the expense of increased computational cost. In the remainder of this section we outline FSIP. In the next section, we describe and extend BS-SIP towards improved computational eficiency.

FSIP associates a variable vector v_0 for the input to the network and constructs the following for each layer f_i:

- Linear symbolic equations $FL_i = C_{FL_i} v_0 + c_{FL_i}$, where $C_{FL_i} \in \mathbb{R}^{s_i \times s_0}$ and $c_{FL_i} \in \mathbb{R}^{s_i}$, expressing lower bounds for the output of the layer. In other words, for every $x_0' \in \mathcal{X}$, we have that $f_i(f_{i-1}(\ldots f_1(x_0')\ldots)) \geq C_{FL_i} x_0' + c_{FL_i}$. Analogous equations $FU_i = C_{FU_i} v_0 + c_{FU_i}$ for the upper bounds are also derived.
- Concrete (i.e., numeric) lower and upper bounds fl_i and fu_i for the output of the layer.

Given an adversarial robustness problem $(f, \mathcal{X}, \mathcal{Y})$, we define in the following the symbolic and concrete bounds for the layers in f, as derived by FSIP.

Initialisation. The bounds for the input to the network are instantiated as prescribed by the verification problem:

$$FL_0 = FU_0 = I_{s_0} v_0, \quad \mathit{fl}_0 = x_0 - \epsilon, \quad \mathit{fu}_0 = x_0 + \epsilon,$$

where I_{s_0} is the identity matrix of size s_0. Having obtained the symbolic bounds $FL_i = C_{FL_i} v_0 + c_{FL_i}$ and $FU_i = C_{FU_i} v_0 + c_{FU_i}$ for the i-th layer, we now show the derivation of (i) the concrete bounds for the layer, (ii) the symbolic bounds of the subsequent layer. The concrete bounds for the layer are computed by instantiating the input variables v_0 with the concrete input bounds fl_0 and fu_0:

$$\mathit{fl}_i = \max(C_{FL_i}, 0)\mathit{fl}_0 + \min(C_{FL_i}, 0)\mathit{fu}_0 + c_{FL_i},$$
$$\mathit{fu}_i = \max(C_{FL_i}, 0)\mathit{fu}_0 + \min(C_{FL_i}, 0)\mathit{fl}_0 + c_{FU_i}.$$

Affine transformation layer. The symbolic bounds for an affine transformation layer f_{i+1} are given by

$$FL_{i+1} = \max(W_{i+1}, 0)FL_i + \min(W_{i+1}, 0)FU_i + b_{i+1},$$
$$FU_{i+1} = \max(W_{i+1}, 0)FU_i + \min(W_{i+1}, 0)FL_i + b_{i+1}.$$

Example 1. *Consider the network from Figure 1. We compute the lower symbolic bounds for layer 1:*

$$FL_1 = \max(W_1, 0)FL_0 + \min(W_1, 0)FU_0$$
$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} I_{s_0} v_0 + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} I_{s_0} v_0 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} v_0.$$

Similarly we can derive that $FU_1 = FL_1$. The lower concrete bounds are derived from FL_1 as follows

$$\mathit{fl}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}.$$

The concrete upper bounds can similarly be computed from FU_1 to obtain $\mathit{fu}_1 = [2 \quad 2]^T$.

ReLU layer. The symbolic bounds for a ReLU layer f_{i+1} are obtained by linearly approximating the ReLU function:

$$FL_{i+1} = \mathit{Diag}(\lambda_{i+1})FL_i,$$
$$FU_{i+1} = \mathit{Diag}(\alpha_i)FU_i - \alpha_i \odot \mathit{fl}_i + \mathsf{ReLU}(\mathit{fl}_i),$$

where $\alpha_i = (\mathsf{ReLU}(\mathit{fu}_i) - \mathsf{ReLU}(\mathit{fl}_i)) \oslash (\mathit{fu}_i - \mathit{fl}_i)$, $\odot$ and $\oslash$ denote Hadamard product and division, and $\lambda_{i+1} \in \mathbb{R}^{s_{i+1}}$ is such that $\lambda_{i+1}(k) = 1$ if the k-th ReLU neuron is *stable*, i.e., it is either *strictly active* ($\mathit{fl}_i(k) \geq 0$) or *strictly inactive* ($\mathit{fu}_i(k) \leq 0$), and $\lambda_{i+1}(k) \in [0, 1]$ if the k-th ReLU neuron is *unstable*, i.e., $\mathit{fl}_i(k) < 0$ and $\mathit{fu}_i(k) > 0$. The vector λ_{i+1} expresses the relaxation slopes for the unstable ReLU neurons which are typically optimised via gradient descent (Xu et al. 2021). Figure 2 graphically depicts the linear relaxation of a ReLU neuron.

Example 2. *Consider again the network from Figure 1. The symbolic bounds for layer 2 are*

$$FL_2 = \mathbf{0}, \textit{ for } \lambda_2 = [0 \quad 0]^T, \textit{ and}$$
$$FU_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} FU_1 - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \odot \begin{bmatrix} -2 \\ -2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} v_0 + \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

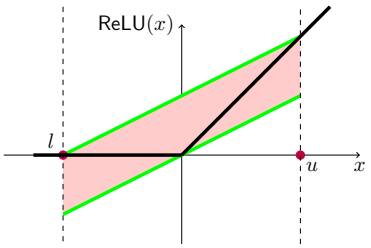


Figure 2: Linear relaxation of the ReLU function $\mathsf{ReLU(x)} = \max(\mathsf{x}, 0)$.

These are used to compute the symbolic bounds for layer 3:

$$FL_3 = \begin{bmatrix} 0 & 0 \\ -0.5 & 0.5 \end{bmatrix} v_0 + \begin{bmatrix} 0 \\ -1 \end{bmatrix}, FU_3 = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \end{bmatrix} v_0 + \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

which in turn give the concrete bounds $\mathit{fl}_3 = [0 \quad -2]^T$ *and* $\mathit{fu}_3 = [3 \quad 2]^T$ *for layer 3. This implies that the first ReLU neuron of layer 4 is stable, so only the second ReLU is relaxed for layer 4:*

$$FL_4 = \mathbf{0}, \quad FU_4 = \begin{bmatrix} 1 & 0 \\ 0.25 & 0.25 \end{bmatrix} v_0 + \begin{bmatrix} 2 \\ 1.5 \end{bmatrix},$$

where we used $\lambda_4 = [1 \quad 0]^T$*, and from where we obtain* $\mathit{fl}_4 = [0 \quad 0]^T$, $\mathit{fu}_4 = [3 \quad 2]^T$.

We conclude this section by noting that the derived concrete bounds fl_L and fu_L for the output of the network can be used to conclude the satisfaction of the adversarial robustness property whenever the lower bound $\mathit{fl}_L(c)$ of the neuron for the class c of the input is greater than the upper bounds of the neurons for all other classes.

3 Dynamic Back-substitution-based SIP

We present Dynamic Back-substitution-based SIP (DBS-SIP), a novel extension of back-substitution-based SIP (BS-SIP) that exhibits improved efficiency without incurring any precision loss. The novel elements of DBS-SIP consist in the dynamic concretisation of the symbolic bounds towards removing redundant computations that impact neither the correctness nor the precision of the procedure.

We begin by associating a vector of variables v_i with the output x_i of each layer f_i. Similarly to FSIP, DBS-SIP iteratively computes linear symbolic equations $BL_{i,0} = C_{BL_{i,0}} v_0 + c_{BL_{i,0}}$ and $BU_{i,0} = C_{BU_{i,0}} v_0 + c_{BU_{i,0}}$ for the lower and upper bounds of the output of each layer f_i, which can be concretised to obtain the concrete lower and upper bounds bl_i and bu_i. The key difference is that whereas FSIP computes the symbolic bounds for a layer by applying the operation the layer implements to the symbolic bounds from the previous layer, DBS-SIP expresses the bounds for a layer as symbolic equations over variables for the outputs of the previous layer. These variables are back-substituted with equations from the previous layers until the equations refer only to variables for the input to the network. This back-substitution process aims at accounting for inter-layer dependencies, thereby facilitating the derivation of tighter bounds, albeit with increased computational cost (Singh et al. 2019b; Zhang et al. 2018). In contrast to previous work in back-substitution-based SIP, DBS-SIP uses the network's ReLU activation patterns to cut down the computational cost of the back-substitution step. This is implemented via the fast-propagation step discussed below. While activation patterns have long been used in linear programming- and SDP-based relaxations, as well as mixed integer linear programming formulations, to simplify the verification problem (Singh et al. 2019b; Botoeva et al. 2020; Weng et al. 2018; Batten et al. 2021), this is, to the best of our knowledge, the first time that they are used towards improving the efficacy of symbolic interval propagation.

DBS-SIP only computes input bounds for the unstable ReLU neurons in the hidden layers since no precision gains are achievable for stable neurons. These bounds are used to derive the linear relaxations of the neurons. While traversing the network backwards to the first layer as part of the back-substitution process, the method may identify additional stable ReLU neurons during the computations executed for each layer. The remaining back-substitution steps required to reach the first layer do not need to be run for any newly stabilised neurons, thereby reducing the computational cost. Having derived all ReLU relaxations, thus built a linear description of the network, the method finally computes the bounds for the output layer using back-substitution.

We now give a formal description of the method. We fix an affine transformation layer f_i and use it to discuss the computation of the lower bounds of f_i — the computation of the upper bounds can be analogously described. DBS-SIP assumes that the symbolic bounds from FSIP have been pre-computed. In the following, we write $BL_{i,j} = C_{BL_{i,j}} v_j + c_{BL_{i,j}}$ to express the lower symbolic bounds of layer f_i over variables for the output of layer f_j. DBS-SIP starts with expressing the bounds of f_i as symbolic equations over the variables v_{i-1} for the output of the previous layer: $BL_{i,i-1} = W_i v_{i-1} + b_i$.

Example 3. *We use the network from Figure 1 to exemplify DBS-SIP. We compute the lower symbolic bounds of layer 5. The bounds are initialised as* $BL_{5,4} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} v_4$.

Following the initialisation of the symbolic bounds $BL_{i,i-1}$, DBS-SIP substitutes the variables v_{i-1} in $BL_{i,i-1}$ with the symbolic equations $BL_{i-1,i-2}$ of layer f_{i-1} over the variables for the output of layer f_{i-2}, and so on, until the symbolic bounds $BL_{i,0}$ are computed. Having obtained the symbolic bounds $BL_{i,j}$ for f_i over the variables v_j for the output of the j-th layer, we now show the derivation of the symbolic bounds $BL_{i,j-1}$ of f_i over the variables v_{j-1} for the output of the $(j-1)$-th layer. This involves a *fast-propagation* step and a *back-substitution* step.

Fast-propagation. Towards reducing the computational cost of back-substituting $BL_{i,j}$, DBS-SIP fast-propagates $BL_{i,j}$ to the input layer using the FSIP symbolic bounds FL_j and FU_j already computed for f_j:

$$\Delta_{i,j} = \max(C_{BL_{i,j}}, 0) FL_j + \min(C_{BL_{i,j}}, 0) FU_j + c_{BL_{i,j}}.$$

It then instantiates $\Delta_{i,j}$ with the input bounds to derive concrete bounds $\delta_{i,j}$ for f_i. These bounds are generally tighter than the concrete bounds generated by FSIP, but typically looser than the bounds obtained by DBS-SIP. This is because the back-substitution process has been partially applied down to layer f_j, thereby capturing inter-layer dependencies between the layers $f_j, \ldots, f_i$, which are ommitted by FSIP, but accounted for by BS-SIP. DBS-SIP uses these bounds of intermediate precision to reduce the number of symbolic equations that are back-substituted from layer f_j to earlier layers. It removes in particular every k-th equation for which we have that $\delta_{i,j}(k) \geq 0$. This condition implies stability for the k-th ReLU neuron of layer f_{i+1}, thus the neuron can be precisely described linearly, hence the bounds for its input need not to be computed. The reduced symbolic bounds, denoted $\overline{BL}_{i,j}$, equal $\overline{BL}_{i,j} = \overline{C}_{BL_{i,j}} v_j + \overline{c}_{BL_{i,i}} = \left[C_{BL_{i,j}}(k,:) \mid \delta_{i,j}(k) < 0\right] v_j + \left[c_{BL_{i,j}}(k) \mid \delta_{i,j}(k) < 0\right]$.

Example 4. *We fast-propagate the symbolic bounds $BL_{5,4}$ from the running example:*

$$\begin{aligned}\Delta_{5,4} &= \max(C_{BL_{5,4}}, 0) FL_4 + \min(C_{BL_{5,4}}, 0) FU_4 \\ &= \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{0} + \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 \\ 0.25 & 0.25 \end{bmatrix} v_0 + \begin{bmatrix} 2 \\ 1.5 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 & 0 \\ -1.25 & -0.25 \end{bmatrix} v_0 + \begin{bmatrix} 0 \\ -3.5 \end{bmatrix}.\end{aligned}$$

The concretisation of $\Delta_{5,4}$ using the input bounds gives $\delta_{5,4} = [0 \quad -5]^T$. This implies stability for the first ReLU neuron of layer 6, so the symbolic bounds $BL_{5,4}$ are reduced to $\overline{BL}_{5,4} = [-1 \quad -1]\, v_4$.

Back-substitution. Following the reduction $\overline{BL}_{i,j}$ of the symbolic bounds $BL_{i,j}$, DBS-SIP substitutes the variables v_j in $\overline{BL}_{i,j}$ with the symbolic lower and upper bounds $BL_{j,j-1}$ and $BU_{j,j-1}$ to derive the symbolic bounds $\overline{BL}_{i,j-1}$ of layer f_i over the variables v_{j-1} for the output of layer f_{j-1}. We have that $BL_{i,j-1} = \max(C_{BL_{i,j}}, 0) BL_{j,j-1} + \min(C_{BL_{i,j}}, 0) BU_{j,j-1} + c_{BL_{i,j}}$, where $BL_{j,j-1}$ is given for affine and ReLU layers as:

- *Affine layer.* $BL_{j,j-1} = BU_{j,j-1} = W_j v_{j-1} + b_j$.
- *ReLU layer.* $BL_{j,j-1} = Diag(\lambda_j) v_{j-1}$, and $BU_{j,j-1} = Diag(\alpha_{j-1}) v_{j-1} - \alpha_{j-1} \odot bl_{j-1} + \mathsf{ReLU}(bl_{j-1})$, where $\alpha_{j-1} = \frac{\mathsf{ReLU}(bu_{j-1}) - \mathsf{ReLU}(bl_{j-1})}{bu_{j-1} - bl_{j-1}}$.

Example 5. *We back-substitute the reduced lower symbolic bounds $\overline{BL}_{5,4} = [-1 \quad -1]\, v_4$ from the running example to derive $BL_{5,3}$. We have that $BL_{4,3} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} v_3$ and $BU_{4,3} = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} v_3 + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, where we have used that $\lambda_4 = [1 \quad 0]^T$, $bl_3 = [0 \quad -2]^T$ and $bu_3 = [3 \quad 2]^T$. Therefore, $BL_{5,3} = [0 \quad 0]\, BL_{4,3} + [-1 \quad -1]\, BU_{4,3} = [-1 \quad -0.5]\, v_3 - 1$.*

Overhead and governance heuristic. While DBS-SIP reduces the size of the symbolic bounds to be back-substituted, it also induces the overhead of the application of the fast-propagation step, which substitutes the variables of DBS-SIP with the FSIP symbolic bounds. To reduce this overhead we develop a heuristic that compares, at each back-substitution step, the expected computational savings resulting from the reduction of the symbolic bounds and the expected computational cost of fast-propagation.

To estimate the computational savings from the reduction of the symbolic bounds $BL_{i,j}$, we concretise $BL_{i,j}$ using the concrete bounds bl_j and bu_j of the j-th layer to obtain $\gamma = \max(BL_{i,j}, 0) bl_j + \min(BL_{i,j}, 0) bu_j$. The number of ReLU neurons that can be proven stable using these bounds equals $\eta = \sum \mathbf{1}_{\gamma \geq 0}$. This number expresses a lower bound for the number of ReLU neurons that can be shown stable using fast-propagation. The savings of reducing the symbolic bounds are estimated therefore as the cost of back-substituting η equations layer-by-layer down to the input layer. This equals $\mathcal{O}(\eta s_j s_{j-1} + \ldots + \eta s_1 s_0)$, where each term $\eta s_k s_{k-1}$ refers to the cost of multiplying the coefficient matrix of the symbolic equations of the η neurons over the variables for the k-th layer (of size $\eta \times s_k$) with the coefficient matrix of $BL_{k,k-1}$ of size $s_k \times s_{k-1}$.

The computational cost of performing fast propagation is estimated on the other hand as $\mathcal{O}(s_i s_j s_0)$. This refers to the cost of multiplying the coefficient matrix of $BL_{i,j}$ of size $s_i \times s_j$ with the coefficient matrix of FL_j of size $s_j \times s_0$.

This estimated cost of the fast-propagation step and estimated savings of the reduction of the symbolic bounds indicate that DBS-SIP is more performant for low-dimensional inputs, as the next section empirically shows.

4 Evaluation

To evaluate the proposed approach we implement the standard back-substitution algorithm (Zhang et al. 2018; Singh et al. 2019b) and the dynamic back-substitution algorithm as described in the previous section. The implementation is in Python and uses PyTorch (Paszke et al. 2019) for efficient vectorised computations and GPU acceleration. The experiments are conducted on one server and one workstation. The server is running Fedora 35 equipped with an AMD EPYC 7453 28-Core Processor and 512GB of RAM. The workstation is running Ubuntu 22 equipped with an AMD Ryzen Threadripper 3970X 32-Core Processor, 256 GB RAM, and a NVIDIA GeForce RTX 3090 GPU.

We compare the performance of the algorithms on two benchmarks from the Verification of Neural Networks Competitions (VNNComp): "cifar_biasfield" from the 2022 competition (Müller et al. 2022) and "TinyYOLO" from the 2023 competition (Brix et al. 2023). The "cifar_biasfield" benchmark consists of one fully convolutional network with 8 layers, 363k parameters and 45k nodes. The "TinyYOLO" benchmark consists of a downsized version of the YOLO model with 14 layers and 92k parameters. We further run experiments with the pre-trained ResNet18 (11.7m parameters) and ResNet34 (21.8m parameters) from PyTorch's TorchVision package version 0.16.0. For the TorchVision and TinyYOLO models we considered brightness perturbations of the inputs (corresponding to $\mathcal{X}$ in Definition 1) (Kouvaros et al. 2023). For the cifar_biasfield benchmark, we used 3-order multiplicative bias fields as described in (Henriksen et al. 2021). For the ResNet benchmarks, we

Model	Perturbation Type	ϵ	#verified	#undecided	t_{base}	t_{nhdyn}	Δ_{nhdyn} (%)	t_{dyn}	Δ_{dyn} (%)
TinyYOLO	brightness	10^{-3}	72	0	14682	-	-	14027	-4.46
		10^{-4}	72	0	3016	-	-	1470	-51.25
		10^{-5}	72	0	426	-	-	418	-1.87
		10^{-6}	72	0	138	-	-	137	-0.72
cifar_biasfield	bias fields	$6 \cdot 10^{-2}$†	0	72	1708	1843	7.85	1883	10.20
		10^{-2}	0	72	1114	1227	10.18	1316	18.19
		10^{-3}	72	0	329	337	2.46	277	-15.98
		10^{-4}	72	0	100	100	0.87	95	-4.15
		10^{-5}	72	0	89	89	0.06	89	0
ResNet18	brightness	10^{-2}	0	50	34374	35854	4.30	33572	-2.33
		10^{-3}	14	36	23347	22975	-1.59	19559	-16.22
		10^{-4}	39	11	16056	14578	-9.21	9976	-37.87
		10^{-5}	38	12	11643	11106	-4.61	7152	-38.57
ResNet34	brightness	10^{-2}	0	50	103177	104327	1.11	101702	-1.43
		10^{-3}	0	50	74598	74733	0.18	70811	-5.08
		10^{-4}	50	0	52851	52923	0.14	40243	-23.85
		10^{-5}	50	0	46238	46827	1.27	29866	-35.41

Table 1: Experimental Evaluation on incomplete verification (single back-substitution passes). $t_{\text{base}}, t_{\text{nhdyn}}$ and t_{dyn} denote the runtime of standard back-substitution, dynamic back-substitution without our heuristic and dynamic back-substitution with our heuristic, respectively. Negative Δ values indicate lower runtimes for dynamic back-substitution compared to standard back-substitution. Dagger (†): Original perturbation size used in VNNComp 2022.

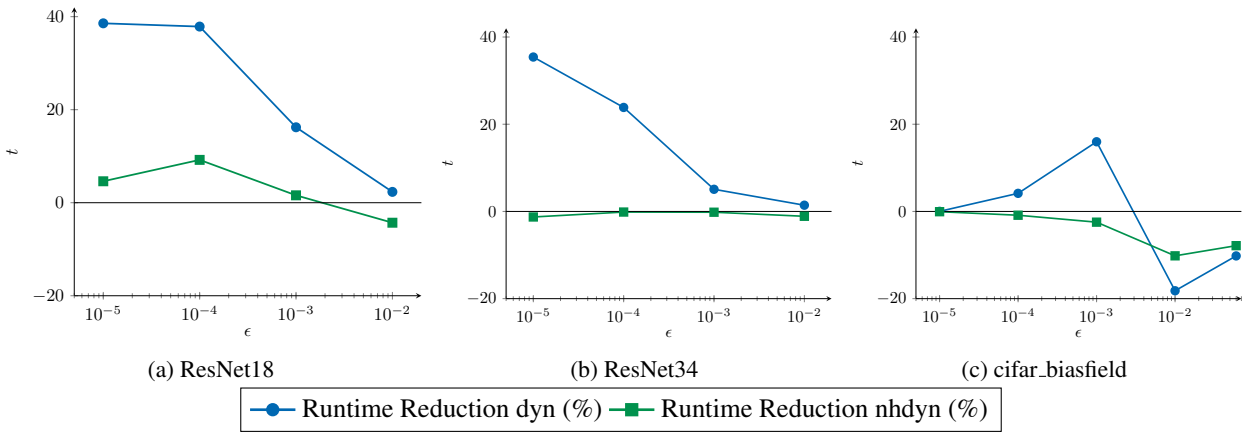
used 50 randomly generated verification properties. For the other benchmarks, we used the properties provided as part of the official VNNComp repository.

Experimental results on back-substitution passes. The results obtained from running back-substitution passes are presented in Table 1. We show the runtime of standard back-substitution as t_{base} and that of DBS-SIP as t_{dyn}. We compute the percentage difference in runtime as $\Delta = \frac{t_{\text{dyn}} - t_{\text{base}}}{t_{\text{base}}}$. For small perturbation sizes, DBS-SIP yields significant runtime reductions. The results indicate that the method stabilises neurons at intermediate layers and, thus, simplifies the bounding matrices. Consequentially, the computational costs are reduced, which explains the improved runtimes. For instance, in the case of TinyYOLO, the runtime is halved for $\epsilon = 10^{-4}$. For the ResNet models, we observe similar speed-ups of up to 40% for small epsilons. However, for large perturbation sizes, we observe smaller gains or even increased runtimes (see, for instance, cifar_biasfield with $\epsilon \geq 10^{-2}$). This is expected since large perturbation sizes may result in wider bounds, hence reducing the number of stable ReLUs, thus potentially making the overhead of DBS-SIP to outweigh the benefit of ReLU stabilisation.

We note that the relative benefit of DBS-SIP also decreases for the smallest perturbation sizes for the TinyYOLO and cifar_biasfield models. This may be a consequence of the forward pass already identifying most ReLUs as stable due to tighter bounds, thus eliminating the need for back-substitution passes altogether. In contrast, for the ResNets, we observe that the gains are still significant for the smallest perturbation sizes as the networks are deeper and larger. Their size generally leads to looser bounds, resulting in fewer stable neurons being identified by the forward pass.

Ablation study. Table 1 presents the results of an ablation study for the evaluation of the heuristic introduced in Section 3 that governs the application of DBS-SIP. The study was conducted on a small (cifar_biasfield), medium (ResNet18), and large (ResNet34) model. We compare the performance of DBS-SIP with the heuristic enabled against a version that attempts to stabilise all unstable nodes at every back-substitution step. We report the runtime of DBS-SIP without the heuristic as t_{nhdyn} alongside the previously introduced t_{base} and t_{dyn}. For the comparison, we compute the percentage runtime difference achieved by DBS-SIP without the heuristic and visualise the results in Figure 3. In line with our discussion in Section 3, we find that the heuristic is essential for DBS-SIP to provide any gains. Without the heuristic, we observe no gains for the cifar_biasfield benchmarks, significantly smaller gains for the ResNet18, and no gains for the ResNet34. The results clearly demonstrate the added benefit of the proposed heuristic.

Experimental results on complete verification runs. To assess the performance of our method in a complete verification setting, we extended the verifier Venus (Kouvaros and Lomuscio 2021) with our implementation of DBS-SIP. We only run our experiments for a medium (cifar_biasfield) and large (ResNet18) model since the incomplete verifier evaluated before already verifies all samples for the smaller TinyYOLO model. The results of the experiments are presented in Table 2 where we find that the performance gains reach 16.50% for $\epsilon = 10^{-3}$ for the medium-sized cifar_biasfield and up to 27% for the large ResNet18. These gains appear to be lower than those for incomplete verifica-



(a) ResNet18 (b) ResNet34 (c) cifar_biasfield

Figure 3: Ablation Study. Positive values correspond to a reduction in runtime which is computed as Reduction $= \frac{t_{\text{base}} - t_{\text{dyn}}}{t_{\text{base}}}$.

Model	Perturbation	ϵ	ver/to	t_{base}	t_{dyn}	Δ_{dyn}
cifar_biasfield	bias fields	10^{-2}	68/4	34275	34516	0.70
		10^{-3}	72/0	333	278	-16.50
		10^{-4}	72/0	99	96	-3.17
		10^{-5}	72/0	88	89	1.40
ResNet18	brightness	10^{-2}	0/50	68458	69354	-0.58
		10^{-3}	34/16	28140	26807	-3.19
		10^{-4}	39/11	9974	8044	-7.11
		$10^{-5\dagger}$	38/7	9243	6915	-27.29

Table 2: Experimental evaluation on complete verification via the Venus verifier. t_{base} and t_{dyn} denote the runtime of standard and dynamic back-substitution with our heuristic, respectively. "ver" and "to" stand for the number of verified/timed out queries. Dagger (†): Five cases in which splitting ran out of memory are excluded from the results.

tion because of the increased number of timeouts encountered which increase the runtime for verification with and without DBS-SIP and reduce the relative difference between the two. When these timeouts are not accounted for, the performance gains for DBS-SIP are of similar magnitude to the case of incomplete verification.

Similarly to incomplete verification, we notice a slight increase in runtime in cases where no neurons can be stabilised because of large perturbation sizes (e.g. for cifar_biasfield with $\epsilon = 10^{-2}$) or small perturbation sizes where most ReLUs are already stabilised with standard SIP (e.g. cifar_biasfield with $\epsilon = 10^{-5}$). The increase in runtime for these cases is the result of the overhead of computing the dynamic back-substitution heuristic.

Summary. The empirical evaluation shows that DBS-SIP can significantly reduce the runtime of back-substitution. In our ablation study we further show that heuristically determining good candidates for ReLU stabilisation is key to achieving gains.

5 Conclusions

Bound-propagation-based methods have had a significant impact towards scaling neural network verification to large models. The methods remain the core components for both incomplete and complete verification frameworks. Improving their efficiency is therefore an important step to validate even deeper and larger models, such those operating in the computer vision domain.

In this paper we made a contribution towards this direction. We introduced dynamic back-substitution, a novel bound-propagation-based method that reduces the computations required by state-of-the-art approaches while not incurring any precision loss on the output bounds. The method was empirically shown to often require less time than the state-of-the-art to resolve verification queries. As it relies on dynamic inference of ReLU stability at every step of the back-substitution process, the method naturally targets performance improvements for low-dimensional specifications. This is confirmed by our experimental results.

To control its effective usage, we introduced a heuristic based on the concretisation of the symbolic bounds at each back-substitution step. While the overhead of computing the heuristic results in little to no gains for small models, the heuristic significantly reduces the required runtime in many cases of practical interest and results in up to 40% performance gains when compared to the state-of-the-art on large models such as ResNet34.

Beside the results reported on VENUS, we expect the advances here proposed on dynamic back-substitution to improve the performance of other neural network verifiers that employ a back-substitution routine.

Acknowledgements

Benedikt Brückner acknowledges partial support from UKRI via the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (EP/S023356/1). Alessio Lomuscio acknowledges partial support from the Royal Academy of Engineering via a Chair in Emerging Technologies.

References

Anderson, G.; Pailoor, S.; Dillig, I.; and Chaudhuri, S. 2019. Optimization and Abstraction: A Synergistic Approach for Analyzing Neural Network Robustness. In *40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI19)*, 731–744. ACM Press.

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. 2020. Strong mixed-integer programming formulations for trained neural networks. In *Integer Programming and Combinatorial Optimization*, volume 11480 of *LNCS*, 27–42. Springer.

Balunovic, M.; Baader, M.; Singh, G.; Gehr, T.; and Vechev, M. 2019. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS19)*, 15313–15323. Curran Associates, Inc.

Bastani, O.; Ioannou, Y.; Lampropoulos, L.; Vytiniotis, D.; Nori, A.; and Criminisi, A. 2016. Measuring Neural Net Robustness with Constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS16)*, 2613–2621.

Batten, B.; Kouvaros, P.; Lomuscio, A.; and Zheng, Y. 2021. Efficient Neural Network Verification via Layer-based Semidefinite Relaxations and Linear Cuts. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2184–2190. ijcai.org.

Batten, B.; Zheng, Y.; Palma, A. D.; Kouvaros, P.; and Lomuscio, A. 2024. Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI24)*, 2362–2369.

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of Neural Networks via Dependency Analysis. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI20)*, 3291–3299. AAAI Press.

Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2023. The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results. *arXiv preprint arXiv:2312.16760*.

Bunel, R.; Turkaslan, I.; Torr, P.; Kohli, P.; and Mudigonda, P. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS18)*, 4790–4799. Curran Associates, Inc.

Cheng, C.; Nührenberg, G.; and Ruess, H. 2017. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA17)*, 251–268. Springer.

Dvijotham, K.; Stanforth, R.; Gowal, S.; Mann, T.; and Kohli, P. 2018. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*.

Ehlers, R. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA17)*, volume 10482 of *Lecture Notes in Computer Science*, 269–286. Springer.

Fazlyab, M.; Morari, M.; and Pappas, G. J. 2020. Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming. *IEEE Transactions on Automatic Control*.

Gowal, S.; Dvijotham, K.; Stanforth, R.; Bunel, R.; Qin, C.; Uesato, J.; Arandjelovic, R.; Mann, T.; and Kohli, P. 2019. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. *arXiv preprint arXiv:1810.12715*.

Hashemi, V.; Kouvaros, P.; and Lomuscio, A. 2021. OSIP: Tightened Bound Propagation for the Verification of ReLU Neural Networks. In *Proceedings of the 19th International Conference on Software Engineering and Formal Methods (SEFM21)*, 463–480. IEEE Computer Society.

Henriksen, P.; Hammernik, K.; Rueckert, D.; and Lomuscio, A. 2021. Bias Field Robustness Verification of Large Neural Image Classifiers. In *Proceedings of the 32nd British Machine Vision Conference (BMVC21)*. BMVA Press.

Henriksen, P.; and Lomuscio, A. 2021. DEEPSPLIT: an Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2549–2555. ijcai.org.

Henriksen, P.; and Lomuscio, A. 2023. Robust Training of Neural Networks against Bias Field Perturbations. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI23)*, 14865–14873. AAAI Press.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17)*, volume 10426 of *Lecture Notes in Computer Science*, 3–29. Springer.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17)*, volume 10426 of *Lecture Notes in Computer Science*, 97–117. Springer.

Katz, G.; Huang, D.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV19)*, 443–452.

Kouvaros, P.; Kyono, T.; Leofante, F.; Lomuscio, A.; Margineantu, D.; Osipychev, D.; and Zheng, Y. 2021. Formal Analysis of Neural Network-Based Systems in the Aircraft Domain. In *Proceedings of the 24th International Symposium on Formal Methods (FM21)*, volume 13047 of *Lecture Notes in Computer Science*, 730–740. Springer.

Kouvaros, P.; Leofante, F.; Chung, C.; Edwards, B.; Margineantu, D.; and Lomuscio, A. 2023. Verification of Semantic Key Point Detection for Aircraft Pose Estimation. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR23)*, 757–762. ijcai.org.

Kouvaros, P.; and Lomuscio, A. 2018. Formal Verification of CNN-based Perception Systems. *arXiv preprint arXiv:1811.11373*.

Kouvaros, P.; and Lomuscio, A. 2021. Towards Scalable Complete Verification of ReLU Neural Networks via Dependency-based Branching. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2643–2650. ijcai.org.

Lan, J.; Brueckner, B.; and Lomuscio, A. 2023. A Semidefinite Relaxation Based Branch-and-Bound Method for Tight Neural Network Verification. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI23)*, 14946–14954. AAAI Press.

Lan, J.; Zheng, Y.; and Lomuscio, A. 2022. Tight Neural Network Verification via Semidefinite Relaxations and Linear Reformulations. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI22)*. AAAI Press.

Lan, J.; Zheng, Y.; and Lomuscio, A. 2023. Iteratively Enhanced Semidefinite Relaxations for Efficient Neural Network Verification. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI23)*, 14937–14945. AAAI Press.

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

Mirman, M.; Gehr, T.; and Vechev, M. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML18)*, 3575–3583. PMLR.

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

Palma, A. D.; Bunel, R.; Desmaison, A.; Dvijotham, K.; Kohli, P.; Torr, P.; and Kumar, M. P. 2021. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS19)*, 8024–8035. Curran Associates, Inc.

Raghunathan, A.; Steinhardt, J.; and Liang, P. S. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Proceedings of the 32th International Conference on Neural Information Processing Systems (NIPS18)*, 10900–10910.

Singh, G.; Ganvir, R.; Püschel, M.; and Vechev, M. 2019a. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems (NeurIPS19)*, 15098–15109. Curran Associates, Inc.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018a. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS18)*, 10802–10813.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018b. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems (NeurIPS18)*, 10802–10813. Curran Associates, Inc.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019b. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL): 41.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019c. Boosting Robustness Certification of Neural Networks. In *International Conference on Learning Representations (ICLR19)*. OpenReview.net.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR14)*.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR19)*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS18)*, 6367–6377. Curran Associates, Inc.

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 Security Symposium (USENIX18)*.

Wang, S.; Zhang, H.; Xu, K.; Lin, X.; Jana, S.; Hsieh, C.; and Kolter, J. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*.

Weng, T.; Zhang, H.; Chen, H.; Song, Z.; Hsieh, C.; Boning, D.; Dhillon, I.; and Daniel, L. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML18)*.

Xu, K.; Zhang, H.; Wang, S.; Wang, Y.; Jana, S.; Lin, X.; and Hsieh, C.-J. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *Proceedings of the 9th International Conference on Learning Representations (ICLR21)*. OpenReview.net.

Zhang, H.; Wang, S.; Xu, K.; Li, L.; Li, B.; Jana, S.; Hsieh, C.-J.; and Kolter, J. Z. 2022. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS22)*, 1656–1670. Curran Associates, Inc.

Zhang, H.; Weng, T.; Chen, P.; Hsieh, C.; and Daniel, L. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS18)*, 4944–4953.