Formal Verification for Neural Networks with General Nonlinearities via Branch-and-Bound

Zhouxing Shi [1] **Qirui Jin** [2] **Huan Zhang** [3] **Zico Kolter** [3] **Suman Jana** [4] **Cho-Jui Hsieh** [1]

Abstract

Bound propagation with branch-and-bound (BaB) is so far among the most effective methods for neural network (NN) verification. However, existing works with BaB have mostly focused on NNs with piecewise linear activations only, especially ReLU neurons. In this paper, we develop a framework for conducting BaB based on bound propagation with general branching points and an arbitrary number of branches, as an important move for extending verification to models with various nonlinearities beyond ReLU. Our framework strengthens verification for common *element-wise* activation functions, as well as other *multi-dimensional* nonlinear operations that arise naturally in computation graphs, such as multiplication and division. In addition, we find that existing branching heuristics for choosing neurons to branch for ReLU networks are insufficient for general nonlinearities, and we design a new heuristic named BBPS, which outperforms the heuristic obtained by directly extending the existing ones originally developed for ReLU networks. We empirically demonstrate the effectiveness of our BaB framework on verifying a wide range of NNs, including networks with Sigmoid, Tanh, or Sin activations, LSTMs, and ViTs, which have various nonlinearities.

1. Introduction

Neural network (NN) verification plays a crucial role in formally verifying whether a neural network satisfies specific properties, such as safety or robustness, prior to its deployment in safety-critical applications. Mathematically, verifiers aim to compute bounds on the output neurons within a pre-defined input region. As computing exact bounds is NP-complete (Katz et al., 2017) even for simple ReLU networks, it becomes crucial to relax the bound computation process to improve efficiency. Bound propagation methods (Wang et al., 2018b; Wong & Kolter, 2018; Zhang et al., 2018; Dvijotham et al., 2018; Henriksen & Lomuscio, 2020; Singh et al., 2019) are commonly used, which relax nonlinearities in neural networks into linear lower and upper bounds that can be efficiently propagated. The linear relaxation relies on intermediate layer bounds, which are recursively computed through bound propagation. However, if the intermediate bounds are not sufficiently tight, the relaxation often results in loose output bounds, particularly for deeper networks.

To further tighten the bounds for bound propagation, Branch-and-Bound (BaB) has been utilized in many works (Bunel et al., 2018; 2020; Xu et al., 2021; Lu & Mudigonda, 2020; De Palma et al., 2021; Wang et al., 2021; Ferrari et al., 2021). BaB iteratively branches the intermediate bounds so that the original verification is branched into subdomains with tighter intermediate bounds. Subsequently, these subdomains can be bounded individually, leading to tighter linear relaxations. However, previous works mostly focused on ReLU networks due to the simplicity of ReLU from its piecewise linear nature. Branching a ReLU neuron only requires branching at 0, and it immediately becomes linear in either branch around 0. Conversely, handling neural networks with nonlinearities beyond ReLU, such as LSTMs (Hochreiter & Schmidhuber, 1997) and Transformers (Vaswani et al., 2017) which also have nonlinearities beyond activation functions such as multiplication and division, introduces additional complexity as the convenience of piecewise linearity diminishes. While there are existing works considering BaB for NNs beyond ReLU networks, they often specialize in specific types of nonlinearities, such as (Henriksen & Lomuscio, 2020; Wu et al., 2022) focusing on S-shaped activations exclusively. A more principled framework for handling general nonlinearities is lacking, leaving ample room for further advancements in verifying non-ReLU networks.

In this paper, we propose a principled verification framework with BaB for neural networks with general nonlinearities. We generalize the α,β-CROWN verifier (Xu et al., 2020; 2021; Wang et al., 2021) which is based on linear

[1]University of California, Los Angeles [2]University of Michigan [3]Carnegie Mellon University [4]Columbia University. Correspondence to: Zhouxing Shi <z.shi@ucla.edu>.

Presented at the 2^{nd} Workshop on Formal Verification of Machine Learning, co-located with the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA., 2023.

bound propagation but initially restricted to BaB for ReLU activations. We resolve multiple challenges to enable BaB for general nonlinearities beyond piecewise linear ReLU. Specifically, we formulate the BaB problem under the linear bound propagation framework in a more general manner. This formulation encompasses general branching points (in contrast to simply 0 for ReLU) and a general number of branches (in contrast to two branches for ReLU which naturally has two pieces).

We encode such general branching into constraints in linear bound propagation by optimizable Lagrange multipliers to improve the verified bounds. Moreover, we find that the existing branching heuristic named "BaBSR" for selecting neurons to branch from works for ReLU networks (Bunel et al., 2020) is suboptimal for networks with general nonlinearities. It is because while making an estimation on the impact of a branching will have on the output bounds, for their convenience BaBSR discards an important term which is found to be negligible on ReLU networks yet we find to be important on general nonlinearities in this paper. Thereby, to improve the effectiveness of BaB, we introduce a new branching heuristic named "Branching via Bound Propagation with Shortcuts (BBPS)" with a more accurate estimation by carefully leveraging the linear bounds from bound propagation.

We demonstrate the effectiveness of the new framework on a variety of networks, including feedforward networks with Sigmoid, Tanh, or Sin activations, LSTMs, Vision Transformers (ViTs). These networks involve various nonlinearities including S-shaped activations, periodic trigonometric functions, and also multiplication and division which are multi-dimensional nonlinear operations beyond activation functions. Our BaB is generally effective and outperforms the existing baselines.

2. Background

The NN verification problem. Let $f : \mathbb{R}^d \mapsto \mathbb{R}^K$ be a neural network taking input $\mathbf{x} \in \mathbb{R}^d$ and outputting $f(\mathbf{x}) \in \mathbb{R}^K$. Suppose $\mathcal{C}$ is the input region to be verified, and $s : \mathbb{R}^K \mapsto \mathbb{R}$ is an output specification function, $h : \mathbb{R}^d \mapsto \mathbb{R}$ is the function that combines the NN and the output specification as $h(\mathbf{x}) = s(f(\mathbf{x}))$. Then NN verification can typically be formulated as verifying if $h(\mathbf{x}) > 0, \forall \mathbf{x} \in \mathcal{C}$ provably holds. A commonly adopted special case is robustness verification given a small input region, where $f(\mathbf{x})$ is a K-way classifier and $h(\mathbf{x}) \coloneqq \min_{i \neq c}\{f_c(\mathbf{x}) - f_i(\mathbf{x})\}$ checks the worst-case margin between the ground-truth class c and any other class i, and if $h(\mathbf{x}) > 0, \forall \mathbf{x} \in \mathcal{C}$, the classifier is provably robust for any input within $\mathcal{C}$. In particular, the input region is often taken as a small ℓ_∞-ball around a data point $\mathbf{x}_0$, i.e., $\mathcal{C} \coloneqq \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_0\|_\infty \leq \epsilon\}$ where ϵ is the radius of the ℓ_∞ ball and $(\mathbf{x} - \mathbf{x}_0)$ is a perturbation around $\mathbf{x}_0$. Robustness verification under an ℓ_∞ input region is a succinct and useful problem for provably verifying the robustness properties of a model and also benchmarking NN verifiers, although there are other NN verification problems beyond robustness. We also focus on this setting for its simplicity following prior works.

Linear bound propagation based on α,β-CROWN . We develop our new framework based on α,β-CROWN (Xu et al., 2020; 2021; Wang et al., 2021) which is among the state-of-the-art NN verifiers (Bak et al., 2021; Müller et al., 2022a) yet it conducts BaB for ReLU only and has limited support for nonlinearities other than ReLU. α,β-CROWN is based on linear bound propagation (Zhang et al., 2018) which computes a sound lower bound for $h(\mathbf{x})$ by propagating linear bounds w.r.t. the output of one or more intermediate layers as

$$h(\mathbf{x}) \geq \sum\nolimits_i \mathbf{A}_i \hat{\mathbf{x}}_i + \mathbf{c}, \tag{1}$$

where $\hat{\mathbf{x}}_i$ $(i \leq n)$ is the output of layer i in the network with n layers, $\mathbf{A}_i$ are the coefficients w.r.t. intermediate layer i in the linear bound, and $\mathbf{c}$ is a bias term dynamically accumulated during the bound propagation. In the beginning, the linear bound is trivially $h(\mathbf{x}) \geq \mathbf{I} \cdot h(\mathbf{x}) + \mathbf{0}$ which is actually an equality but is still consistent with Eq. (1). During the bound propagation, $\mathbf{A}_i \hat{\mathbf{x}}_i$ in Eq. (1) is recursively substituted by the linear bound of $\hat{\mathbf{x}}_i$ w.r.t its input. For simplicity, suppose layer $i-1$ is the input to layer i and $\hat{\mathbf{x}}_i = h_i(\hat{\mathbf{x}}_{i-1})$, where $h_i(\cdot)$ is the computation for layer i. And suppose we have the linear bounds of $\hat{\mathbf{x}}_i$ w.r.t its input $\hat{\mathbf{x}}_{i-1}$ as:

$$\underline{\mathbf{a}}_i \hat{\mathbf{x}}_{i-1} + \underline{\mathbf{b}}_i \leq \hat{\mathbf{x}}_i = h_i(\hat{\mathbf{x}}_{i-1}) \leq \overline{\mathbf{a}}_i \hat{\mathbf{x}}_{i-1} + \overline{\mathbf{b}}_i, \tag{2}$$

with parameters $\underline{\mathbf{a}}_i, \underline{\mathbf{b}}_i, \overline{\mathbf{a}}_i, \overline{\mathbf{b}}_i$ for the linear bounds, and "$\leq$" holds elementwise. Then $\mathbf{A}_i \hat{\mathbf{x}}_i$ can be substituted and lower bounded by:

$$\mathbf{A}_i \hat{\mathbf{x}}_i \geq \mathbf{A}_{i-1} \hat{\mathbf{x}}_{i-1} + \big(\mathbf{A}_{i,+} \underline{\mathbf{b}}_i + \mathbf{A}_{i,-} \overline{\mathbf{b}}_i\big), \tag{3}$$

$$\text{where} \quad \mathbf{A}_{i-1} = \big(\mathbf{A}_{i,+} \underline{\mathbf{a}}_i + \mathbf{A}_{i,-} \overline{\mathbf{a}}_i\big), \tag{4}$$

"+" and "-" in the subscripts denote taking positive and negative elements respectively, and in this way the linear bounds are propagated from layer i to layer $i-1$. Ultimately the linear bounds can be propagated to the input of the network $\mathbf{x}$ as $h(\mathbf{x}) \geq \mathbf{A}_0 \mathbf{x} + \mathbf{c}$, $\mathbf{A}_0 \in \mathbb{R}^{1 \times d}$, where the input can be viewed as the 0-th layer. Depending on $\mathcal{C}$, this linear bound can be concretized into a lower bound without $\mathbf{x}$. If $\mathcal{C}$ is an ℓ_∞ ball, we have

$$\forall \|\mathbf{x} - \mathbf{x}_0\|_\infty \leq \epsilon, \quad \mathbf{A}_0 \mathbf{x} + \mathbf{c} \geq \mathbf{A}_0 \mathbf{x}_0 - \epsilon \|\mathbf{A}_0\|_1 + \mathbf{c}. \tag{5}$$

To construct Eq. (2), if $h_i(\cdot)$ is inherently linear, its linear bounds are simply itself. Otherwise, linear relaxation is used, and it relaxes a nonlinearity to lower and upper

bound the nonlinearity by linear functions. An intermediate bound on $\hat{\mathbf{x}}_{i-1}$ as $\mathbf{l}_{i-1} \leq \hat{\mathbf{x}}_{i-1} \leq \mathbf{u}_{i-1}$ is usually required for the relaxation, which can be obtained by running additional bound propagation and treating the intermediate layers as the output of a network. For example, if $h_i(\hat{\mathbf{x}}_{i-1})$ is a ReLU activation, linear relaxation is needed for the j-th neuron only if $\mathbf{l}_{i-1,j} < 0 < \mathbf{u}_{i-1,j}$ as otherwise the function is linear. The upper bound can be a line connecting endpoints $(\mathbf{l}_{i-1,j}, h_i(\mathbf{l}_{i-1,j}))$ and $(\mathbf{u}_{i-1,j}, h_i(\mathbf{u}_{i-1,j}))$, and the lower bound can be any $\alpha\hat{\mathbf{x}}_{i-1,j} (0 \leq \alpha \leq 1)$. Linear relaxation can contain optimizable parameters to tighten the bounds (Lyu et al., 2020; Xu et al., 2021), and we use $\boldsymbol{\alpha}$ to denote all the optimizable parameters in the linear relaxation.

Branch-and-Bound (BaB). BaB has been widely applied to tighten the bounds. Each time it *branches* the intermediate bound of a selected neuron j in a selected layer $i-1$, $\hat{\mathbf{x}}_{i-1,j} \in [\mathbf{l}_{i-1,j}, \mathbf{u}_{i-1,j}]$, into smaller subdomains with tighter intermediate bounds. Then BaB *bounds* such subdomain respectively and take the worst bound from the subdomains as the new bound. This process is repeated iteratively to gradually improve the bounds. α,β-CROWN only conducts BaB for ReLU neurons. For a selected neuron with intermediate bounds $\hat{\mathbf{x}}_{i,j} \in [\mathbf{l}_{i-1,j}, \mathbf{u}_{i-1,j}]$ satisfying $\mathbf{l}_{i-1,j} < 0 < \mathbf{u}_{i-1,j}$, the bounds can be branched into two subdomains as $\hat{\mathbf{x}}^{(1)}_{i-1,j} \in [\mathbf{l}^{(1)}_{i-1,j}, \mathbf{u}^{(1)}_{i-1,j}]$ and $\hat{\mathbf{x}}^{(2)}_{i-1,j} \in [\mathbf{l}^{(2)}_{i-1,j}, \mathbf{u}^{(2)}_{i-1,j}]$ with $\mathbf{u}^{(1)}_{i-1,j} = \mathbf{l}^{(2)}_{i-1,j} = 0$, and thereby the ReLU neuron in each subdomain immediately becomes linear and the linear relaxation becomes exact. For each ReLU branching, α,β-CROWN further encodes a constraint $\hat{\mathbf{x}}^{(1)}_{i-1,j} \leq 0$ or $\hat{\mathbf{x}}^{(2)}_{i-1,j} \geq 0$ for the two subdomains respectively by a Lagrangian multiplier denoted as $\beta^{(1)}_{i-1,j}$ or $\beta^{(2)}_{i-1,j}$, where $\beta^{(1)}_{i-1,j}, \beta^{(2)}_{i-1,j} \geq 0$. Specifically, it adds a term $\beta^{(1)}_{i-1,j}\hat{\mathbf{x}}^{(1)}_{i-1,j}$ or $-\beta^{(2)}_{i-1,j}\hat{\mathbf{x}}^{(2)}_{i-1,j}$ respectively to the right-hand-side (RHS) of Eq. (4) and this extra term can be combined with $\mathbf{A}_{i-1}\hat{\mathbf{x}}_{i-1}$. We use $\boldsymbol{\beta}$ to denote all such Lagrangian multipliers.

Overall, α,β-CROWN uses linear bound propagation with linear relaxation enhanced by optimizable parameters $\boldsymbol{\alpha}$, and it uses BaB to tighten the bounds with branching constraints encoded with parameters $\boldsymbol{\beta}$. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ parameters are optimized by gradient descent. BaB in α,β-CROWN is restricted to ReLU and is not applicable to more general nonlinearities.

3. Method

3.1. Overall Framework

Notations. We have reviewed α,β-CROWN in Section 2, where we only considered a feedforward NN for its simplicity. But the linear bound propagation technique has been generalized to general computational graphs to support various NN architectures (Xu et al., 2020). In our method, we also consider a general computational graph $h(\mathbf{x})$ for input region $\mathbf{x} \in \mathcal{C}$. Instead of a feedforward network with n *layers* in Section 2, we consider a computational graph with n *nodes*, where each node i computes some function $h_i(\cdot)$ that may either correspond to a linear layer in the NN or a nonlinearity. We use $\hat{\mathbf{x}}_i$ to denote the output of node i which may contain many neurons, and we use $\hat{\mathbf{x}}_{i,j}$ to denote the output of the j-th neuron in node i. Intermediate bounds of node i may be needed to relax and bound $h_i(\cdot)$, and we use $\mathbf{l}_{i,j}, \mathbf{u}_{i,j}$ to denote the intermediate lower bound and upper bound respectively. We use $\mathbf{l}$ and $\mathbf{u}$ to denote all the intermediate lower bounds and upper bounds respectively for the entire computational graph.

Initial verification. Before entering BaB, we first compute initial verified bounds by bound propagation with optimizable linear relaxation. Specifically, we use $V_\alpha(h, \mathcal{C}, \boldsymbol{\alpha})$ to denote the linear bound propagation-based verifier with $\boldsymbol{\alpha}$ denoting all the parameters in the optimizable relaxation, and we compute initial verified bounds by optimizing $\boldsymbol{\alpha}$, as $h(\mathbf{x}) \geq \max_{\boldsymbol{\alpha}} V_\alpha(h, \mathcal{C}, \boldsymbol{\alpha})$ $(\forall \mathbf{x} \in \mathcal{C})$, where $\boldsymbol{\alpha}$ is constrained within a domain that ensures the soundness of the relaxation, such as $0 \leq \alpha \leq 1$ for ReLU mentioned in Section 2. All the intermediate bounds are also updated with the updating $\boldsymbol{\alpha}$, and we obtain the optimized intermediate bounds $\mathbf{l}, \mathbf{u}$. The verification finishes if $V_\alpha(h, \mathcal{C}, \boldsymbol{\alpha}) > 0$ holds already. Since α,β-CROWN has limited support on nonlinearities beyond ReLU, we have derived new optimizable linear relaxation we encounter, as will be discussed in Section 3.5.

Branch-and-Bound. Otherwise, we enter our BaB to tighten the bounds. We maintain a dynamic pool of intermediate bound domains, $\mathcal{D} = \{(\mathbf{l}^{(i)}, \mathbf{u}^{(i)})\}_{i=1}^m$, where $m = |\mathcal{D}|$ is the number of current domains, and initially $\mathcal{D} = \{(\mathbf{l}, \mathbf{u})\}$ with the intermediate bounds from the initial verification. In each iteration of BaB, we pick a domain that leads to the worst verified bounds, and for the picked domain, we select a neuron to branch and obtain new subdomains. For the new subdomains, we update $\mathbf{l}, \mathbf{u}$ for the branched neurons, and we also use $\boldsymbol{\beta}$ parameters for the Lagrange multipliers in the branching constraints. For each new subdomain, given updated $\mathbf{l}, \mathbf{u}$ and the parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$, we denote a verified lower bound computed during BaB as $V(h, \mathbf{l}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, and we optimize $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to obtain an optimized lower bound for $h(\mathbf{x})$:

$$h(\mathbf{x}) \geq \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} V(h, \mathbf{l}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \ \ \forall \mathbf{x} \in \mathcal{C}. \tag{6}$$

Subdomains with $V(h, \mathbf{l}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) > 0$ are verified and discarded, otherwise they are added to $\mathcal{D}$ for further branching. In our implementation, a batch of multiple domains can be branched in parallel. We repeat the process until there is no

domain left in $\mathcal{D}$ and the verification succeeds, or when the timeout is reached and the verification fails. We illustrate the framework in Figure 1.

3.2. Branching for General Nonlinearities

Branching on ReLU networks as studied by prior works is a special case of branching on general nonlinearities. For ReLU networks, branching is needed only if $\mathbf{l}_{i,j} < 0 < \mathbf{u}_{i,j}$ for a neuron, and the only reasonable way is to branch at 0 and split the intermediate bounds into two branches, $[\mathbf{l}_{i,j}, 0]$ and $[0, \mathbf{u}_{i,j}]$, so that ReLU is linear for both sides, as shown in Figure 2a. However, branching for general nonlinearities on general computational graphs is more complex. First, branching can be needed even if $\mathbf{u}_{i,j} \leq 0$ or $\mathbf{l}_{i,j} \geq 0$ and it requires considering branching at points other than 0. As shown in Figure 2b, a Sigmoid activation is still nonlinear when given intermediate bound $[0.25, 1.75]$, and it cannot be branched at 0 which is out of $[0.25, 1.75]$. But it can be branched at a point within $[0.25, 1.75]$ such as 1.0. Second, unlike ReLU, general nonlinearities usually do not consist of two linear pieces, and the intermediate bounds may be branched into more than two branches at once, as shown in Figure 2c. Third, unlike typical activation functions, some nonlinearities may take more than one input. For example, there may be a node computing $\hat{\mathbf{x}}_i = h_i(\hat{\mathbf{x}}_{i-1}, \hat{\mathbf{x}}_{i-2}) = \hat{\mathbf{x}}_{i-1}\hat{\mathbf{x}}_{i-2}$, as appeared in Transformers (Vaswani et al., 2017; Shi et al., 2019) or LSTMs (Hochreiter & Schmidhuber, 1997; Ko et al., 2019). The multiplication between $\hat{\mathbf{x}}_{i-1}$ and $\hat{\mathbf{x}}_{i-2}$ is generally a nonlinear function unless one of $\hat{\mathbf{x}}_{i-1}$ and $\hat{\mathbf{x}}_{i-2}$ is constant and does not depend on $\mathbf{x}$. For such nonlinearities, there are multiple input nodes that can be branched. Fourth, on general computational graphs, a node can also be followed by multiple nonlinearities, as appeared in LSTMs, and then branching intermediate bounds of this node can affect multiple nonlinearities.

To resolve these challenges, we propose a new and more general formulation for branching on general nonlinearities for general computational graphs. Each time we consider branching the intermediate bounds of a neuron j in a node i, namely $[\mathbf{l}_{i,j}, \mathbf{u}_{i,j}]$, if node i is the input of some nonlinearity. We consider branching the concerned neuron into K branches with branching points $\mathbf{p}_{i,j}^{(1)}, \cdots, \mathbf{p}_{i,j}^{(K-1)}$, and then the intermediate bounds become:

$$[\mathbf{l}_{i,j}, \mathbf{u}_{i,j}] \rightarrow [\mathbf{l}_{i,j}, \mathbf{p}_{i,j}^{(1)}], [\mathbf{p}_{i,j}^{(1)}, \mathbf{p}_{i,j}^{(2)}], \cdots, [\mathbf{p}_{i,j}^{(K-1)}, \mathbf{u}_{i,j}], \tag{7}$$

for the K branches respectively. In this work, we instantiate Eq. (7) as uniformly branching $[\mathbf{l}_{i,j}, \mathbf{u}_{i,j}]$ into K branches where we take $K = 3$ for non-ReLU models.

We select the neuron to branch by a heuristic to approximately maximize the bound improvement after the branching, as discussed in Section 3.4. If neuron j in node i is selected, we use the new intermediate bounds of each branch to update the linear relaxation of the impacted nonlinearities. We also add branching constraints parameterized by $\boldsymbol{\beta}$, as will be discussed in Section 3.3. Then compute new verified bounds for the branches by solving Eq. (6) with multiple iterations optimizing $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

Note that in our formulation, we consider each node that is *the input to some nonlinearities* and decide if we branch on this node, and it allows us to naturally generalizes to nonlinearities with multiple input nodes as well as multiple nonlinearities sharing the the input node. It would be more convenient and general compared to considering *the nonlinearities themselves*, and how all the input nodes of a nonlinearity shall be branched, yet the input nodes may be shared by some other nonlinearities.

3.3. Encoding the General Branching Constraints

We encode general branching constraints into the linear bound propagation by $\boldsymbol{\beta}$ Lagrange multipliers which have shown to be important for linear bound propagation in BaB (Wang et al., 2021) which focused on ReLU. For each neuron j in a node i branched as Eq. (7), we obtain branching constraints for the output $\hat{\mathbf{x}}_{i,j}^{(1)}, \cdots, \hat{\mathbf{x}}_{i,j}^{(K)}$ in the K branches respectively: $\hat{\mathbf{x}}_{i,j}^{(1)} - \mathbf{p}_{i,j}^{(1)} \leq 0$, $\hat{\mathbf{x}}_{i,j}^{(2)} - \mathbf{p}_{i,j}^{(2)} \leq 0$, $\mathbf{p}_{i,j}^{(1)} - \hat{\mathbf{x}}_{i,j}^{(2)} \leq 0, \cdots, \mathbf{p}_{i,j}^{(K-1)} - \hat{\mathbf{x}}_{i,j}^{(K)} \leq 0$. These constraints can be encoded into the bound propagation by Lagrangian multipliers, and we add $\mathbf{s}_{i,j}^{(k)}$ for the k ($1 \leq k \leq K$)-th branch. For $2 \leq k \leq K-1$, we have

$$\mathbf{s}_{i,j}^{(k)} \coloneqq \beta_{i,j}^{(k,1)}(\hat{\mathbf{x}}_{i,j}^{(k)} - \mathbf{p}_{i,j}^{(k)}) + \beta_{i,j}^{(k,2)}(\mathbf{p}_{i,j}^{(k-1)} - \hat{\mathbf{x}}_{i,j}^{(k)}),$$

and in we also have $\mathbf{s}_{i,j}^{(1)} \coloneqq \beta_{i,j}^{(1)}(\hat{\mathbf{x}}_{i,j}^{(1)} - \mathbf{p}_{i,j}^{(1)})$ and $\mathbf{s}_{i,j}^{(K)} \coloneqq \beta_{i,j}^{(K)}(\mathbf{p}_{i,j}^{(K-1)} - \hat{\mathbf{x}}_{i,j}^{(K)})$. With the Lagrangian multipliers, the linear bound during the linear bound propagation for each branch from Eq. (1) can be updated into:

$$h(\mathbf{x}) \geq \sum\nolimits_i \left(\mathbf{A}_i \hat{\mathbf{x}}_i + \sum\nolimits_j \mathbf{s}_{i,j} \right) + \mathbf{c} = \sum\nolimits_i \tilde{\mathbf{A}}_i \hat{\mathbf{x}}_i + \tilde{\mathbf{c}}, \tag{8}$$

where $\mathbf{s}_{i,j}$ denotes all the added terms with Lagrangian multipliers, $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{c}}$ are the updated parameters in the linear bound after merging $\mathbf{A}_i$ with coefficients in every $\mathbf{s}_{i,j}$ and merging $\mathbf{c}$ with the biases in every $\mathbf{s}_{i,j}$ respectively. Since this is done for each of the K branches, we omit the superscripts "(k)" here. Then the linear bound propagation can continue with the updated parameters.

3.4. A New Branching Heuristic for General Nonlinear Functions

In each branching iteration, we aim to pick some neuron j in node i on which the branching potentially leads to the

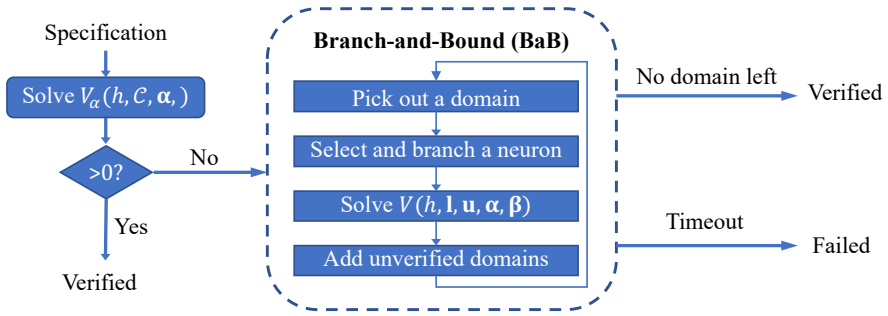
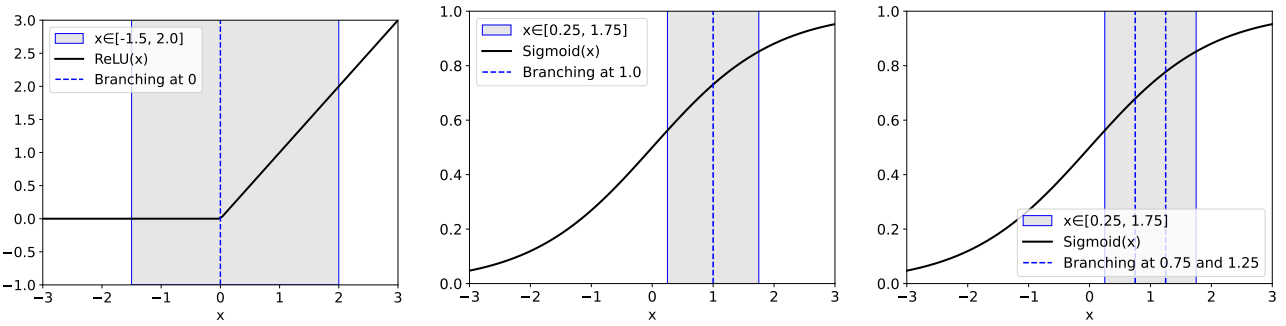


Figure 1. Illustration of our framework, as described in Section 3.1.



(a) Branching a neuron with ReLU activation.

(b) Branching a neuron with Sigmoid activation into two branches.

(c) Branching a neuron with Sigmoid activation into three branches.

Figure 2. Illustration of branching the intermediate bound of a neuron with different activations.

largest improvement on the verified bounds:

$$\arg\max_{i,j} \min_{1\le k\le K} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} V(h, \underline{B}(\mathbf{l}, i, j, k), \overline{B}(\mathbf{u}, i, j, k), \boldsymbol{\alpha}, \boldsymbol{\beta}), \tag{9}$$

where we use $\underline{B}(\mathbf{l}, i, j, k)$ to denote the updated intermediate lower bounds for the k-th branch after branching neuron j in node i, and similarly $\overline{B}(\mathbf{u}, i, j, k)$ for the updated intermediate upper bounds. Previous works typically use some branching heuristic (Bunel et al., 2018; 2020; Lu & Mudigonda, 2020; De Palma et al., 2021) which approximates the potential improvement in an efficient way.

Suppose we consider branching a neuron j in node i and we aim to estimate $V(\cdot)$ in Eq. (9) for each branch k. In linear bound propagation, when the bounds are propagated to node i, we have:

$$\begin{aligned} h(\mathbf{x}) &\ge \mathbf{A}_{i,j}^{(k)} \hat{\mathbf{x}}_{i,j} + \mathbf{c}^{(k)} \\ &\ge V(h, \underline{B}(\mathbf{l}, i, j, k), \overline{B}(\mathbf{u}, i, j, k), \boldsymbol{\alpha}, \boldsymbol{\beta}), \end{aligned}$$

where we use $\mathbf{A}_{i,j}^{(k)}$ and $\mathbf{c}^{(k)}$ to denote the parameters in the linear bounds for the k-th branch. Note that branching a neuron in node i only affects the linear relaxation of nonlinear nodes immediately after node i (i.e., output nodes of i), and thus $\mathbf{A}_{i,j}^{(k)}$ and $\mathbf{c}^{(k)}$ can be computed by only propagating the linear bounds from the output nodes of i using stored linear bounds rather than from the ultimate output of $h(\mathbf{x})$. If we want to exactly obtain $V(h, \underline{B}(\mathbf{l}, i, j, k), \overline{B}(\mathbf{u}, i, j, k), \boldsymbol{\alpha}, \boldsymbol{\beta})$, then we need to further propagate the linear bounds until the input of the network, which is costly.

For a more efficient estimation, the BaBSR heuristic (Bunel et al., 2020) originally for ReLU networks essentially propagates the bounds only to the node before the branched one with an early stop, as they then ignore the coefficients ($\mathbf{A}_{i-1,j}^{(k)}$ for a feedforward NN) without propagating further. Note that we have described this heuristic in a general way, although it was originally for ReLU networks only. We call it "BaBSR-like" as a direct adaption from BaBSR (Bunel et al., 2020). However, we find a BaBSR-like branching heuristic is suboptimal on the models with general nonlinearities we experimented, as the heuristic ignores the important impact of the discarded coefficients on the verified bounds.

In this work, we propose a new branching heuristic named Branching via Bound Propagation with Shortcuts (BBPS), where we use a shortcut to directly propagate the bounds to the input. We expect it to more precisely estimate the potential improvement than simply discarding terms during the bound propagation, and more efficient than simply propagate the bounds layer by layer to the input. Specifically,

we save the linear bounds of all the potentially branched intermediate layers during the initial verification before BaB. For every neuron j in intermediate layer i, we record:

$$\forall \mathbf{x} \in \mathcal{C}, \quad \underline{\hat{\mathbf{A}}}_{ij}\mathbf{x} + \underline{\hat{\mathbf{c}}}_{ij} \leq \hat{\mathbf{x}}_{ij} \leq \overline{\hat{\mathbf{A}}}_{ij}\mathbf{x} + \overline{\hat{\mathbf{c}}}_{ij}, \tag{10}$$

where $\underline{\hat{\mathbf{A}}}_{ij}, \underline{\hat{\mathbf{c}}}_{ij}, \overline{\hat{\mathbf{A}}}_{ij}, \overline{\hat{\mathbf{c}}}_{ij}$ are parameters for the linear bounds. These are obtained when linear bound propagation is used for computing the intermediate bounds $[\mathbf{l}_{i,j}, \mathbf{u}_{i,j}]$ and the linear bounds are propagated to the input $\mathbf{x}$. We then use Eq. (10) to compute a lower bound for $\mathbf{A}^{(k)}_{i,j}\hat{\mathbf{x}}_{i,j} + \mathbf{c}^{(k)}$:

$$\begin{aligned}
&\mathbf{A}^{(k)}_{i,j}\hat{\mathbf{x}}_{i,j} + \mathbf{c}^{(k)} \\
\geq\ &(\mathbf{A}^{(k)}_{i,j,+}\underline{\hat{\mathbf{A}}}_{ij} + \mathbf{A}^{(k)}_{i,j,-}\overline{\hat{\mathbf{A}}}_{ij})\mathbf{x} \\
&+ \mathbf{A}^{(k)}_{i,j,+}\underline{\hat{\mathbf{c}}}_{ij} + \mathbf{A}^{(k)}_{i,j,-}\overline{\hat{\mathbf{c}}}_{ij} + \mathbf{c}^{(k)}, \qquad \forall \mathbf{x} \in \mathcal{C},
\end{aligned}$$

and then the RHS can be concretized by Eq. (5) and serve as an approximation for $V(\cdot)$ after branching. In this way, the linear bounds are directly propagated from node i to input $\mathbf{x}$ and concretized using a shortcut. Utilizing previously saved linear bounds has also been used in previous works (Shi et al., 2019; Zhong et al., 2021) for speeding up bound propagation, while we show that it can serve as a better branching heuristic for general nonlinearities.

3.5. Optimized Linear Relaxation for Less Studied Nonlinearities

We derive new optimized linear relaxation for nonlinearities not supported in α,β-CROWN , including multiplication and trigonometric functions. For multiplication, the unoptimized relaxation in α,β-CROWN is basically from (Shi et al., 2019) originally for verifying Transformers. Shi et al. (2019) mentioned that there are two different relaxations with optimal tightness, and they arbitrarily take one. We optimize the relaxation by interpolating between the two optimal cases with a parameter for the interpolation. We also handle periodic functions such as $\sin$ by optimizing tangent points used for constructing the linear relaxation, where the domain of the tangent points is carefully designed to produce sound bounds when the function can contain multiple periods. We discuss details in Appendix A.

4. Experiments

4.1. Settings

We experiment on NNs with various nonlinearities as shown in **??**. We consider the commonly used ℓ_∞ robustness verification specification on image classification. We compare with baselines (Singh et al., 2019; Müller et al., 2022b; Henriksen & Lomuscio, 2020; Ryou et al., 2021; Bonaert et al., 2021; Wu et al., 2022; Wei et al., 2023) on models they support respectively. We adopt some MNIST (LeCun et al., 2010) models trained from existing works, along with their data instances for verification. We also compute an upper bound on the number of potentially verifiable instances by PGD attack (Madry et al., 2018), as a sound verification should not verify on instances where a PGD attack can successfully discover counterexamples. Besides, we also train several new models on CIFAR-10 (Krizhevsky et al., 2009) by PGD adversarial training (Madry et al., 2018) using an ℓ_∞ perturbation with $\epsilon = 1/255$ in both training and verification. For these CIFAR-10 models, we first run vanilla CROWN (Zhang et al., 2020; Xu et al., 2020) (without $\boldsymbol{\alpha}, \boldsymbol{\beta}$ or BaB) and PGD attack (Madry et al., 2018) on the test set and remove instances on which either PGD attack succeeds or vanilla CROWN can already verify the property. Thereby we only retain instances that can possibly be verified but are relatively hard to verify. If there are more than 100 instances after the filtering, we only retain the first 100 instances. We set a timeout of 300 seconds for our BaB in all the experiments. Details are in Appendix C.

4.2. Experiments on Sigmoid and Tanh networks

MNIST models from Müller et al. (2022b). We first experiment on feedforward Sigmoid networks and Tanh networks for MNIST from Müller et al. (2022b). Table 1 shows the results. On the Sigmoid 9×100 network, we find that the number of verified instances reported by Wu et al. (2022) is particularly high and even exceeds the upper bound by PGD attack, and thus their result may not be fully sound in this case and we skip it in the following comparison. Then, we find that using $\boldsymbol{\alpha}$ only without BaB can already outperforms all the non-CROWN baselines on most of the cases. On 4 out of the 6 models, our BaB with BBPS is able to verify additional instances over using $\boldsymbol{\alpha}$ only. Our method with BaB outperforms all the baselines. We also find that improving on Sigmoid 9×100 and Tanh 6×100 networks by BaB is hard, as the initial bounds are typically too loose on the unverifiable instances, possibly due to these models being trained by standard training without robustness intervention. In Figure 3, we plot the total number of verified instances against the running time for various methods, showing that our method can verify more instances compared to the baselines when the timeout threshold is at least around 10 seconds, and BaB enables us to verify more instances as more time is allowed compared to using $\boldsymbol{\alpha}$ only.

CIFAR-10 models by PGD training. In Table 2, we show results for models on CIFAR-10. The results show that our BaB effectively improves verification beyond using $\boldsymbol{\alpha}$ only without BaB. Besides, the ablation studies show that using the BaBSR-like heuristic adapted from Bunel et al. (2020) as mentioned in Section 3.4 negatively impacts the performance of BaB, providing evidence for the effectiveness of our new branching heuristic. Disabling $\boldsymbol{\beta}$ leads to a reduction in the number of instances, which validates the

Table 1. Number of verified instances out of the first 100 test examples on MNIST for several Sigmoid networks and Tanh networks along with their ϵ from Müller et al. (2022b). "$L \times W$" in the network names denote a fully-connected NN with L layers and W hidden neurons in each layer. The upper bounds in the last row are computed by PGD attack.

Method	Sigmoid Networks			Tanh Networks		
	6×100 $\epsilon=0.015$	6×200 $\epsilon=0.012$	9×100 $\epsilon=0.015$	6×100 $\epsilon=0.006$	6×200 $\epsilon=0.002$	9×100 $\epsilon=0.006$
DeepPoly (Singh et al., 2019)[ab]	30	43	38	38	39	18
PRIMA (Müller et al., 2022b)[a]	53	73	56	61	68	52
VeriNet (Henriksen & Lomuscio, 2020)	65	81	56	31	30	16
Wu et al. (2022)[?]	65	75	96[?]	-	-	-
Vanilla CROWN (Zhang et al., 2018)[b]	53	63	49	18	24	44
α only w/o BaB	62	81	**62**	**65**	72	58
BaB (BBPS)	**71**	**83**	**62**	**65**	**78**	**59**
Upper bound	93	99	92	94	97	96

[a]Results for DeepPoly and PRIMA are directly from Müller et al. (2022b).
[b]While DeepPoly and CROWN are thought to be equivalent on ReLU networks (Müller et al., 2022b), these two works adopt different relaxation for Sigmoid and Tanh, which results in different results here.
[?]For Wu et al. (2022), we found that the result they report on the Sigmoid 9×100 model exceeds the upper bound by PGD attack ($96 > 92$), and thus the result may not be fully valid. Results on Tanh networks are unavailable from Wu et al. (2022).

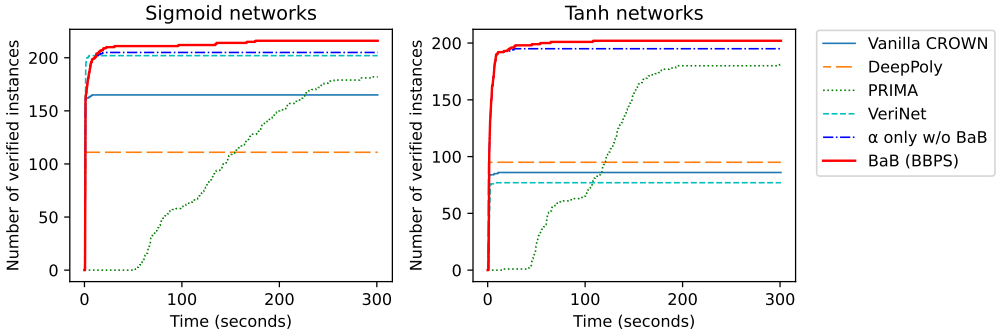


Figure 3. Total number of verified instances against running time threshold, on the three Sigmoid networks (left) and three Tanh networks (right) respectively in Table 1.

Table 2. Number of verified instances out of 100 filtered instances on CIFAR-10 for several Sigmoid networks and Tanh networks with $\epsilon = 1/255$.

Method	Sigmoid Networks				Tanh Networks	
	4×100	4×500	6×100	6×200	4×100	6×100
PRIMA	0	0	0	0	0	0
α only w/o BaB	28	16	43	39	25	6
BaB (BaBSR-like)	34	17	44	41	35	8
BaB (BBPS, w/o β)	47	20	55	47	39	9
BaB (BBPS)	**53**	**21**	**61**	**49**	**41**	**9**

effectiveness of encoding the general branching constraints using β. For the non-CROWN baselines, as we only use relatively hard instances for verification, we find that PRIMA is unable to verify any instance here. On the other hand, VeriNet depends on the FICO Xpress commercial solver which requires a license which we do not have access to[1] for all the models in this experiment due to the problem scale. Thus we omit VeriNet in this experiment.

[1]FICO Xpress declined the request we submitted for the academic license, directing us to obtain it via a (course) tutor, which is not applicable for our research.

4.3. Experiments on LSTMs

Next, we experiment on LSTMs containing more complex nonlinearities, including both Sigmoid and Tanh activations, as well as multiplication as $\text{sigmoid}(x)\tanh(y)$ and $\text{sigmoid}(x)y$. We compare with PROVER (Ryou et al., 2021) which is a specialized verification algorithm for RNN outperforming earlier RNN verification works (Ko et al., 2019). While there are other works on verifying RNN and LSTM, such as (Du et al., 2021; Mohammadinejad et al., 2021; Paulsen & Wang, 2022), we have not found their code, and we also make orthogonal contributions compared them on improving the relaxation for RNN verification. Thus we omit them in our experiments. We take the hardest model, an LSTM for MNIST, from the main experiments of PROVER (other models can be verified by PROVER on more than 90% instances and are thus omitted), where each 28×28 image is sliced into 7 frames for LSTM. We also use two LSTMs trained by ourselves on CIFAR-10, where

we linearly map each 32×32 image into 4 patches as the input tokens, similar to ViTs with patches (Dosovitskiy et al., 2021). Table 3 shows the results. Using $\boldsymbol{\alpha}$ only without BaB can already outperform PROVER with specialized relaxation for RNN and LSTM, and using BaB further boosts the performance.

Table 3. Number of verified instances out of 100 instances on MNIST and CIFAR-10 respectively. The MNIST model follows the setting of the hardest model in the main experiments of PROVER (Ryou et al., 2021) with $\epsilon = 0.01$. The CIFAR-10 models are trained by ourselves with $\epsilon = 1/255$. "LSTM-7-32" indicates an LSTM with 7 input frames and 32 hidden neurons, similar for the other two models.

Method	MNIST Model	CIFAR-10 Models	
	LSTM-7-32	LSTM-4-32	LSTM-4-64
PROVER	63	8	3
$\boldsymbol{\alpha}$ only w/o BaB	83	16	9
BaB (BBPS)	**86**	**25**	**15**
Upper bound	98	100	100

4.4. Experiments on ViTs and Sin Networks

Finally, we also experiment on ViTs and NNs with $\sin$ activation, which contain nonlinearities that are less studied. For ViTs, we compare with DeepT (Bonaert et al., 2021) which is specialized for verifying Transformers without using BaB. We show the results in Table 4, where our methods outperform DeepT and BaB effectively improves the verification. Besides, in Appendix B, we also compare with Wei et al. (2023) which supports verifying attention networks but not the entire ViT, and we experiment on models from Wei et al. (2023), where our methods also outperform Wei et al. (2023). Moreover, we demonstrate our method on networks with $\sin$ activations as periodic activation functions are important for applications such as neural rendering (Sitzmann et al., 2020), and Table 5 shows that our method is also effective.

Table 4. Number of verified instances on ViTs for CIFAR-10 ($\epsilon = 1/255$). There are fewer than 100 instances after the filtering, shown as the upper bounds. "ViT-L-H" stands for L layers and H heads.

Method	ViT-1-3	ViT-1-6	ViT-2-3	ViT-2-6
DeepT	0	1	0	1
$\boldsymbol{\alpha}$ only w/o BaB	1	3	11	7
BaB (BBPS)	**15**	**34**	**28**	**24**
Upper bound	67	92	72	69

5. Related Work

Branch-and-bound (BaB) has been shown to be an effective technique for NN verification (Bunel et al., 2018; Lu & Mudigonda, 2020; Wang et al., 2018a; Xu et al., 2021; De Palma et al., 2021; Kouvaros & Lomuscio, 2021; Wang et al., 2021; Henriksen & Lomuscio, 2021), but most of the existing works focus on ReLU networks and are not directly applicable to networks with nonlinearities beyond ReLU. On BaB for NNs with other nonlinearities, Henriksen & Lomuscio (2020) conducted BaB on Sigmoid and Tanh networks, but their framework still depends on a commercial LP solver which has been argued as less effective than recent NN verification methods using linear bound propagation with branching constraints (Wang et al., 2021). Besides, Wu et al. (2022) studied verifying Sigmoid networks with counter-example-guided abstraction refinement but their method is still specialized for Sigmoid. Moreover, these works have only considered S-shaped activations, and there lacks a general framework supporting general nonlinearities beyond some particular ones, which we address in this paper. Without using BaB, there are also other works studying the relaxation in verifying NNs with various nonlinearities, such as RNNs and LSTMs (Ko et al., 2019; Du et al., 2021; Ryou et al., 2021; Mohammadinejad et al., 2021; Zhang et al., 2023), and also Transformers (Shi et al., 2019; Bonaert et al., 2021; Wei et al., 2023). These works have orthogonal contributions compared to ours using BaB for further improvement above a base verifier. In addition, there are works studying the branching heuristic in verifying ReLU networks, such as filtering initial candidates from BaBSR (Bunel et al., 2020) with a more accurate computation, using Graph Neural Networks for the heuristic (Lu & Mudigonda, 2020), or using a heuristic guided with tighter multiple-neuron relaxation (Ferrari et al., 2021), which may inspire future improvement on the BaB for general nonlinearities.

Table 5. Number of verified instances on $\sin$ networks for CIFAR-10 ($\epsilon = 1/255$).

Method	4×100	4×200	4×500
$\boldsymbol{\alpha}$ only w/o BaB	75	70	59
BaB (BBPS)	**87**	**86**	**73**

6. Conclusions

To conclude, we propose a general BaB framework for NN verification involving general nonlinearities. We also propose a new and more effective branching heuristic for BaB on general nonlinearities and we extend optimized linear relaxation. Experiments on verifying NNs with various nonlinearities demonstrate the effectiveness of our method.

Limitations and Future work. There remain several limitations in this work to be resolved in the future. As mentioned in Section 3.2, we have only used a simple way for deciding the branching points, and it will be interesting for future works to investigate more sophisticated ways.

Besides, for the branching heuristic, we have not added mechanisms such as filtering the candidates after the initial heuristic (De Palma et al., 2021), and we also leave it for future work to study the possibility of applying the latest progress on ReLU networks to strengthen the branching heuristic for general nonlinearities.

References

Bak, S., Liu, C., and Johnson, T. The second international verification of neural networks competition (vnncomp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

Bonaert, G., Dimitrov, D. I., Baader, M., and Vechev, M. Fast and precise certification of transformers. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 466–481, 2021.

Bunel, R., Turkaslan, I., Torr, P. H. S., Kohli, P., and Mudigonda, P. K. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pp. 4795–4804, 2018.

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

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

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Du, T., Ji, S., Shen, L., Zhang, Y., Li, J., Shi, J., Fang, C., Yin, J., Beyah, R., and Wang, T. Cert-rnn: Towards certifying the robustness of recurrent neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pp. 516–534, 2021. ISBN 9781450384544. doi: 10.1145/3460120.3484538.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T. A., and Kohli, P. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pp. 550–559, 2018.

Ferrari, C., Mueller, M. N., Jovanović, N., and Vechev, M. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2021.

Henriksen, P. and Lomuscio, A. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pp. 2513–2520. IOS Press, 2020.

Henriksen, P. and Lomuscio, A. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *IJCAI*, pp. 2549–2555, 2021.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ko, C., Lyu, Z., Weng, L., Daniel, L., Wong, N., and Lin, D. POPQORN: quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3468–3477, 2019.

Kouvaros, P. and Lomuscio, A. Towards scalable complete verification of relu neural networks via dependency-based branching. In *IJCAI*, pp. 2643–2650, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Technical Report TR-2009*, 2009.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Lu, J. and Mudigonda, P. Neural network branching for neural network verification. In *Proceedings of the International Conference on Learning Representations (ICLR 2020)*. Open Review, 2020.

Lyu, Z., Ko, C., Kong, Z., Wong, N., Lin, D., and Daniel, L. Fastened CROWN: tightened neural network robustness certificates. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pp. 5037–5044, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Mohammadinejad, S., Paulsen, B., Deshmukh, J. V., and Wang, C. Diffrnn: Differential verification of recurrent neural networks. In *Formal Modeling and Analysis of Timed Systems: 19th International Conference, FORMATS 2021, Paris, France, August 24–26, 2021, Proceedings 19*, pp. 117–134. Springer, 2021.

Müller, M. N., Brix, C., Bak, S., Liu, C., and Johnson, T. T. The third international verification of neural networks competition (vnn-comp 2022): Summary and results. *arXiv preprint arXiv:2212.10376*, 2022a.

Müller, M. N., Makarchuk, G., Singh, G., Püschel, M., and Vechev, M. Prima: general and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–33, 2022b.

Paulsen, B. and Wang, C. Linsyn: Synthesizing tight linear bounds for arbitrary neural network activation functions. In *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*, pp. 357–376. Springer, 2022.

Ryou, W., Chen, J., Balunovic, M., Singh, G., Dan, A., and Vechev, M. Scalable polyhedral verification of recurrent neural networks. In *International Conference on Computer Aided Verification*, pp. 225–248, 2021.

Shi, Z., Zhang, H., Chang, K.-W., Huang, M., and Hsieh, C.-J. Robustness verification for transformers. In *International Conference on Learning Representations*, 2019.

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

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pp. 6369–6379, 2018a.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018b.

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

Wei, D., Wu, H., Wu, M., Chen, P.-Y., Barrett, C., and Farchi, E. Convex bounds on the softmax function with applications to robustness verification. In *International Conference on Artificial Intelligence and Statistics*, pp. 6853–6878. PMLR, 2023.

Wong, E. and Kolter, J. Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5283–5292, 2018.

Wu, H., Tagomori, T., Robey, A., Yang, F., Matni, N., Pappas, G., Hassani, H., Pasareanu, C., and Barrett, C. Toward certified robustness against real-world distribution shifts. *arXiv preprint arXiv:2206.03669*, 2022.

Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., and Hsieh, C. Automatic perturbation analysis for scalable certified robustness and beyond. In *Advances in Neural Information Processing Systems*, 2020.

Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

Zhang, H., Weng, T., Chen, P., Hsieh, C., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pp. 4944–4953, 2018.

Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D. S., and Hsieh, C. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

Zhang, Y., Du, T., Ji, S., Tang, P., and Guo, S. Rnn-guard: Certified robustness against multi-frame attacks for recurrent neural networks. *arXiv preprint arXiv:2304.07980*, 2023.

Zhong, Y., Ta, Q.-T., Luo, T., Zhang, F., and Khoo, S.-C. Scalable and modular robustness analysis of deep neural networks. In *Programming Languages and Systems: 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17–18, 2021, Proceedings 19*, pp. 3–22. Springer, 2021.

A. Additional Optimizable Linear Relaxation

A.1. Optimizable Linear Relaxation for Multiplication

In this section, we derive our optimizable linear relaxation for multiplication. For each elementary multiplication xy where $x \in [\underline{x}, \overline{x}]$, $y \in [\underline{y}, \overline{y}]$ are the intermediate bounds for x and y, we aim to relax and bound xy as:

$$\forall x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}], \quad \underline{a}x + \underline{b}y + \underline{c} \leq xy \leq \overline{a}x + \overline{b}y + \overline{c}, \tag{11}$$

where $\underline{a}, \underline{b}, \underline{c}, \overline{a}, \overline{b}, \overline{c}$ are parameters in the linear bounds. Shi et al. (2019) derived optimal parameters that minimize the gap between the relaxed upper bound and the relaxed lower bound:

$$\underset{\underline{a},\underline{b},\underline{c},\overline{a},\overline{b},\overline{c}}{\arg\min} \int_{x\in[\underline{x},\overline{x}]} \int_{y\in[\underline{y},\overline{y}]} (\overline{a}x + \overline{b}y + \overline{c}) - (\underline{a}x + \underline{b}y + \underline{c}) \quad \text{s.t. Eq. (11).} \tag{12}$$

However, the optimal parameters they found only guarantee that the linear relaxation is optimal for this node, but not the final bounds after conducting a bound propagation on the entire NN. Therefore, we aim to make these parameters optimizable to tighten the final bounds as previous works did for ReLU networks or S-shaped activations (Xu et al., 2021; Lyu et al., 2020).

We notice that Shi et al. (2019) mentioned that there are two solutions for $\underline{a}, \underline{b}, \underline{c}$ and $\overline{a}, \overline{b}, \overline{c}$ respectively that solves Eq. (12):

$$\begin{cases} \underline{a}_1 = \underline{y} \\ \underline{b}_1 = \underline{x} \\ \underline{c}_1 = -\underline{x}\underline{y} \end{cases}, \quad \begin{cases} \overline{a}_1 = \overline{y} \\ \overline{b}_1 = \underline{x} \\ \overline{c}_1 = -\underline{x}\overline{y} \end{cases}, \tag{13}$$

$$\begin{cases} \underline{a}_2 = \overline{y} \\ \underline{b}_2 = \overline{x} \\ \underline{c}_2 = -\overline{x}\overline{y} \end{cases}, \quad \begin{cases} \overline{a}_2 = \underline{y} \\ \overline{b}_2 = \overline{x} \\ \overline{c}_2 = -\overline{x}\underline{y} \end{cases}. \tag{14}$$

Therefore, to make the parameters optimizable, we introduce parameters $\underline{\alpha}$ and $\overline{\alpha}$, and we interpolate between Eq. (13) and Eq. (14) as:

$$\begin{cases} \underline{a} = \underline{\alpha}\underline{y} + (1-\underline{\alpha})\overline{y} \\ \underline{b} = \underline{\alpha}\underline{x} + (1-\underline{\alpha})\overline{x} \\ \underline{c} = -\underline{\alpha}\underline{x}\underline{y} - (1-\underline{\alpha})\overline{x}\overline{y} \end{cases} \quad \text{s.t. } 0 \leq \underline{\alpha} \leq 1, \tag{15}$$

$$\begin{cases} \overline{a} = \overline{\alpha}\overline{y} + (1-\overline{\alpha})\underline{y} \\ \overline{b} = \overline{\alpha}\underline{x} + (1-\overline{\alpha})\overline{x} \\ \overline{c} = -\overline{\alpha}\underline{x}\overline{y} - (1-\overline{\alpha})\overline{x}\underline{y} \end{cases} \quad \text{s.t. } 0 \leq \overline{\alpha} \leq 1. \tag{16}$$

It is easy to verify that interpolating between two sound linear relaxations satisfying Eq. (11) still yields a sound linear relaxation. And $\underline{\alpha}$ and $\overline{\alpha}$ are part of all the optimizable linear relaxation parameters $\boldsymbol{\alpha}$ mentioned in Section 2.

A.2. Optimizable Linear Relaxation for Sine

We also derive new optimized linear relaxation for periodic functions, in particular $\sin(x)$ for $x \in [\underline{x}, \overline{x}]$. A non-optimizable linear relaxation for $\sin$ already exists in α,β-CROWN and we adopt it as an initialization and focus on making it optimizable. For each of the lower bound and upper bound, the initial relaxation first checks the line connecting $(\underline{x}, \sin(\underline{x}))$ and $(\overline{x}, \sin(\overline{x}))$, and this line is adopted if it is a sound bounding line and no optimization is needed. Otherwise, a tangent line is used as the bounding line, and we optimize the tangent point. Within $[\underline{x}, \overline{x}]$, if $\sin(x)$ happens to be monotonic with at most only one inflection point, the tangent point can be optimized in a way similar to bounding an S-shaped activation (Lyu et al., 2020). Otherwise, the input range contains extreme points, where $\sin(x)$ has extreme points at $\frac{(2k+1)\pi}{2}$ ($k \in \mathbb{Z}$). Suppose $\frac{(2k_1+1)\pi}{2}$ is the extreme point closest to $\underline{x}$ and $\frac{(2k_2+1)\pi}{2}$ is the one closet to $\overline{x}$, within $[\underline{x}, \overline{x}]$. Then we optimize the tangent point within $[\underline{x}, \frac{(2k_1+1)\pi}{2}]$ and $[\frac{(2k_2+1)\pi}{2}, \overline{x}]$ respectively, which yields two set of lower and upper bounds. We compare them with the one connecting $(\underline{x}, \sin(\underline{x}))$ and $(\overline{x}, \sin(\overline{x}))$. We take a sound bounding line that is relatively tighter according to the gap between the bounding line and $\sin(x)$, for the lower bound and upper bound respectively.

B. Additional Results

B.1. Experiments on Convolutional Neural Networks from Müller et al. (2022b)

Table 6. Number of verified instances out of the first 100 test examples on MNIST for convolutional neural networks with Sigmoid and Tanh activations respectively. These models are the "ConvSmall" models in Müller et al. (2022b). VeriNet is not included as it depends on the FICO Xpress solver which requires a non-community license that we cannot obtain (see Section 4.2) for the problem size in these models.

Method	Sigmoid ConvSmall ($\epsilon = 0.014$)	Tanh ConvSmall ($\epsilon = 0.005$)
DeepPoly (Singh et al., 2019)	30	16
PRIMA (Müller et al., 2022b)	51	30
Wu et al. (2022)	63	-
Vanilla CROWN (Zhang et al., 2018)	65	55
$\boldsymbol{\alpha}$ only w/o BaB	84	69
BaB (BBPS)	**92**	**75**
Upper bound	97	98

Table 6 shows results on convolutional neural networks from Müller et al. (2022b) and the conclusions are consistent with those for Table 1.

B.2. Experiments on Self-Attention Networks from (Wei et al., 2023)

To compare with Wei et al. (2023) that only supports verifying single-layer self-attention networks but not the entire ViT, we adopt pre-trained models from (Wei et al., 2023) and run our verification methods under their settings, with 500 test images in MNIST using $\epsilon = 0.02$. We show the results in Table 7, where our methods also outperform (Wei et al., 2023) on all the models.

Table 7. Number of verified instances out of 500 instances in MNIST with $\epsilon = 0.02$. A-small, A-medium and A-big are three self-attention networks with different parameter sizes from (Wei et al., 2023).

Method	A-small	A-medium	A-big
Wei et al. (2023)	406	358	206
$\boldsymbol{\alpha}$ only w/o BaB	444	388	176
BaB (BBPS)	**450**	**455**	**232**
Upper bound	463	479	482

C. Implementation Details

Verification. We implement our verification algorithm based on auto_LiRPA[2] and α,β-CROWN[3], both under the BSD-3-Clause license. We use the Adam optimizer (Kingma & Ba, 2015) to optimize $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ with an initial learning rate of 0.1 and the learning rate is decayed by 2% after each iteration. To solve $V_\alpha(h, \mathcal{C}, \boldsymbol{\alpha})$ in the initial verification, we optimize $\boldsymbol{\alpha}$ for at most 100 iterations. And to solve $V(h, \mathbf{l}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ during BaB, we optimize $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for at most 50 iterations. Our BaB is batched where multiple domains are branched in parallel, and the batch size is dynamic tuned based on the model size to fit the GPU memory.

Training the models. To train our models on CIFAR-10, we use PGD adversarial training (Madry et al., 2018). We use 7 PGD steps during the training and the step size is set to $\epsilon/4$. For training the Sigmoid networks in Table 2, we use the SGD optimizer with a learning rate of 5×10^{-2} for 100 epochs; and for training the Tanh networks, we use the SGD optimizer with a learning rate of 1×10^{-2} for 100 epochs. For training the LSTMs in Table 3, we use the Adam optimizer with a

[2] https://github.com/Verified-Intelligence/auto_LiRPA

[3] https://github.com/Verified-Intelligence/alpha-beta-CROWN

learning of 10^{-3} for 30 epochs. And for training the ViTs, we use the Adam optimizer with a learning of 5×10^{-3} for 100 epochs. For Siren, we use the SGD optimizer with a learning rate of 1×10^{-3} for 100 epochs