Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent works in neural network verification show that cheap incomplete verifiers such as CROWN can be used in Branch-and-Bound (BaB) methods and significantly accelerate complete verification of neural networks. However, they cannot fully handle the neuron split constraints introduced by BaB commonly handled by expensive linear programming (LP) solvers, leading to looser bounds and hurting verification efficiency. In this work, we develop β-CROWN, a new bound propagation based method that can fully encode neuron splits via optimizable parameters $\boldsymbol{\beta}$ constructed from either primal or dual space. When jointly optimized in intermediate layers, β-CROWN generally produces better bounds than typical LP verifiers with neuron split constraints, while being as efficient and parallelizable as CROWN on GPUs. Applied to complete robustness verification benchmarks, β-CROWN with BaB is close to three orders of magnitude faster than LP-based BaB methods, and is at least 3 times faster than winners of VNN-COMP 2020 competition while producing lower timeout rates. By terminating BaB early, our method can also be used for efficient incomplete verification. We achieve higher verified accuracy in many settings over powerful incomplete verifiers, including those based on convex barrier breaking techniques. Compared to the typically tightest but very costly semidefinite programming (SDP) based incomplete verifiers, we obtain higher verified accuracy with three orders of magnitudes less verification time.

1 Introduction

As neural networks (NNs) are being deployed in safety-critical applications, it becomes increasingly important to formally verify their behaviors under potentially malicious inputs. Broadly speaking, the neural network verification problem involves proving certain desired relationships between inputs and outputs (often referred to as *specifications*), such as safety or robustness guarantees, for all inputs inside some domain. Canonically, the problem can be cast as finding the global minima of some functions on the network's outputs (e.g., the difference between the predictions of the true label and another target label), within a bounded input set as constraints. This is a challenging problem due to the non-convexity and high dimensionality of neural networks.

We first focus on complete verification: the verifier should give a definite "yes/no" answer given sufficient time. Many complete verifiers rely on branch and bound (BaB) method [7] involving (1) branching by recursively splitting the original verification problem into subdomains (e.g., splitting a ReLU neuron into positive/negative linear regions) and (2) bounding each subdomain with specialized incomplete verifiers. Traditional BaB-based verifiers use expensive linear programming (LP) solvers [14, 22, 6] as incomplete verifiers which can fully encode neuron split constraints for tight bounding. Meanwhile, a recent work [43] shows that cheap incomplete verifiers can signifi-

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

cantly accelerate complete verifiers on GPUs over LP-based ones thanks to their efficiency. We call these cheap incomplete verifiers *bound propagation methods* [40, 12, 16, 34], i.e., maintaining and propagating tractable and sound bounds through networks, and CROWN [44] is a representative.

However, unlike LP solvers, existing bound propagation methods lack the power to handle neuron split constraints introduced by BaB. For instance, given $x, y \in [-1, 1]$, they can efficiently bound a ReLU's input $x + y$ as $[-2, 2]$ but they have no means to consider neuron split constraints such as $x - y \geq 1$ introduced by splitting another ReLU. Such a problem causes loose bounds and unnecessary branching, hurting the verification efficiency. Even worse, they fail to detect many infeasible subdomains in BaB, leading to incompleteness unless additional checking is performed [43].

In our work, we develop a new, fast bound propagation based incomplete verifier, β-CROWN. It solves an optimization problem equivalent to the expensive LP based methods with neuron split constraints while still enjoying the efficiency of bound propagation methods. β-CROWN contains optimizable parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and any valid settings of these parameters yield sound bounds for verification. These parameters are optimized using a few steps of (super)gradient ascent to achieve bounds as tight as possible. Optimizing $\boldsymbol{\beta}$ can also eliminate many infeasible subdomains and avoid further useless branching. Furthermore, we can jointly optimize $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for intermediate layer bounds, allowing β-CROWN to tighten relaxations and outperform typical LP verifiers with fixed intermediate layer bounds. Unlike traditional LP-based BaB methods, β-CROWN can be efficiently implemented with an automatic differentiation framework on GPUs to fully exploit the power of modern accelerators. The combination of β-CROWN and BaB (β-CROWN BaB) produces a complete verifier with GPU acceleration, reducing the verification time of traditional LP based BaB verifiers [7] by almost *three orders of magnitudes* on standard benchmarking models. Compared to popular verifiers like ERAN and OVAL (the leading methods in VNN competition 2020 [21]), our approach is at least 3 times faster. Compared to all state-of-the-art GPU-based complete verifiers [6, 43, 9, 22, 5] including a concurrent work BaDNB [10], our approach is 2 to 7 times faster with lower timeout rates.

Finally, by terminating our complete verifier β-CROWN BaB early, our approach can also function as a more accurate incomplete verifier by returning an incomplete but sound lower bound of all subdomains explored so far. We achieve better verified accuracy on a few benchmarking models over powerful incomplete verifiers including those based on multi-neuron convex relaxations [33, 35, 26] and semidefinite relaxations [8]. Compared to the typically tightest but very costly incomplete verifier SDP-FO [8] based on tight semidefinite programming (SDP) relaxations [27, 13], our method obtains consistently higher verified accuracy while reducing verification time by three orders of magnitudes.

2 Background

Neural network verification. We define the input of a neural network as $x \in \mathbb{R}^{d_0}$, and define the weights and biases of an L-layer neural network as $\mathbf{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{d_i}$ ($i \in \{1, \cdots, L\}$) respectively. For simplicity we assume that $d_L = 1$ so $\mathbf{W}^{(L)}$ is a vector and $\mathbf{b}^{(L)}$ is a scalar. The neural network function $f : \mathbb{R}^{d_0} \to \mathbb{R}$ is defined as $f(x) = z^{(L)}(x)$, where $z^{(i)}(x) = \mathbf{W}^{(i)} \hat{z}^{(i-1)}(x) + \mathbf{b}^{(i)}$, $\hat{z}^{(i)}(x) = \sigma(z^{(i)}(x))$ and $\hat{z}^{(0)}(x) = x$. σ is the activation function and we use ReLU throughout this paper. When the context is clear, we omit $\cdot(x)$ and use $z_j^{(i)}$ and $\hat{z}_j^{(i)}$ to represent the *pre-activation* and *post-activation* values of the j-th neuron in the i-th layer. Neural network verification seeks the solution of the optimization problem in Eq. 1:

$$\min f(x) := z^{(L)}(x) \quad \text{s.t. } z^{(i)} = \mathbf{W}^{(i)} \hat{z}^{(i-1)} + \mathbf{b}^{(i)}, \hat{z}^{(i)} = \sigma(z^{(i)}), x \in \mathcal{C}, i \in \{1, \cdots, L-1\} \quad (1)$$

The set $\mathcal{C}$ defines the allowed input region and our aim is to find the minimum of $f(x)$ for $x \in \mathcal{C}$, and throughout this paper we consider $\mathcal{C}$ as an ℓ_∞ ball around a data example x_0: $\mathcal{C} = \{x \mid \|x - x_0\|_\infty \leq \epsilon\}$ but other ℓ_p norms can also be supported. In practical settings, we typically have "specifications" to verify, which are (usually linear) functions of neural network outputs describing the desired behavior of neural networks. For example, to guarantee robustness we typically investigate the margin between logits. Because the specification can also be seen as an output layer of NN and merged into $f(x)$ under verification, we do not discuss it in detail in this work. We consider the canonical specification $f(x) > 0$: if we can prove that $f(x) > 0, \ \forall x \in \mathcal{C}$, we say $f(x)$ is verified.

When $\mathcal{C}$ is a convex set, Eq. 1 is still a non-convex problem because the constraints $\hat{z}^{(i)} = \sigma(z^{(i)})$ are non-convex. Given unlimited time, *complete* verifiers can solve Eq. 1 exactly: $f^* = \min f(x), \ \forall x \in \mathcal{C}$, so we can always conclude if the specification holds or not for any problem instance. On the other

hand, *incomplete* verifiers usually relax the non-convexity of neural networks to obtain a tractable lower bound of the solution $\underline{f} \leq f^*$. For cases where $\underline{f} < 0$, we cannot conclude the specification.

The LP Verifier. A commonly used incomplete verification technique is to relax non-convex ReLUs with linear constraints and turn the verification problem into a linear programming (LP) problem, which can then be solved with linear solvers. We refer to it as "LP verifier" in this paper. Specifically, given $\mathrm{ReLU}(z_j^{(i)}) := \max(0, z_j^{(i)})$ and its intermediate layer bounds $\mathbf{l}_j^{(i)} \leq z_j^{(i)} \leq \mathbf{u}_j^{(i)}$, each ReLU can be categorized into three cases linearly relaxed differently: (1) if $\mathbf{l}_j^{(i)} \geq 0$ (ReLU in linear region) then $\hat{z}_j^{(i)} = z_j^{(i)}$; (2) if $\mathbf{u}_j^{(i)} \leq 0$ (ReLU in inactive region) then $\hat{z}_j^{(i)} = 0$; (3) if $\mathbf{l}_j^{(i)} \leq 0 \leq \mathbf{u}_j^{(i)}$ (ReLU is *unstable*) then three linear bounds are used: $\hat{z}_j^{(i)} \geq 0$, $\hat{z}_j^{(i)} \geq z_j^{(i)}$, and $\hat{z}_j^{(i)} \leq \frac{\mathbf{u}_j^{(i)}}{\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}} \left(z_j^{(i)} - \mathbf{l}_j^{(i)} \right)$; they are often referred to as the "triangle" relaxation [14, 40]. The intermediate layer bounds $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$ are usually obtained from a cheaper bound propagation method (see next subsection). LP verifiers can provide relatively tight bounds but linear solvers are still expensive especially when the network is large. Also, unlike our β-CROWN, they have to use fixed intermediate bounds and cannot use the joint optimization of intermediate layer bounds (Section 3.3) to tighten relaxation.

2.1 CROWN: Efficient Incomplete Verification by Propagating Linear Bounds

Another cheaper way to give a lower bound for the objective in Eq. 1 is through sound bound propagation. CROWN [44] is a representative method that propagates a linear bound of $f(x)$ w.r.t. every intermediate layer in a backward manner until reaching the input x. CROWN uses two linear constraints to relax unstable ReLU neurons: a linear upper bound $\hat{z}_j^{(i)} \leq \frac{\mathbf{u}_j^{(i)}}{\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}} \left(z_j^{(i)} - \mathbf{l}_j^{(i)} \right)$ and a linear lower bound $\hat{z}_j^{(i)} \geq \boldsymbol{\alpha}_j^{(i)} z_j^{(i)}$ $(0 \leq \boldsymbol{\alpha}_j^{(i)} \leq 1)$. We can then bound the output of a ReLU layer:

Lemma 2.1 (ReLU relaxation in CROWN)**.** *Given $w, v \in \mathbb{R}^d$, $\mathbf{u} \leq v \leq \mathbf{l}$ (element-wise), we have*

$$w^\top \mathrm{ReLU}(v) \geq w^\top \mathbf{D} v + b',$$

where $\mathbf{D}$ is a diagonal matrix containing free variables $0 \leq \boldsymbol{\alpha}_j \leq 1$ only when $\mathbf{u}_j > 0 > \mathbf{l}_j$ and $w_j \geq 0$, while its rest values as well as constant b' are determined by $\mathbf{l}, \mathbf{u}, w$.

Detailed forms of each term are listed in Appendix A. Lemma 2.1 can be repeatedly applied, resulting in an efficient back-substitution procedure to derive a linear lower bound of NN output w.r.t. x:

Lemma 2.2 (CROWN bound [44])**.** *Given an L-layer ReLU NN $f(x) : \mathbb{R}^{d_0} \to \mathbb{R}$ with weights $\mathbf{W}^{(i)}$, biases $\mathbf{b}^{(i)}$, pre-ReLU bounds $\mathbf{l}^{(i)} \leq z^{(i)} \leq \mathbf{u}^{(i)}$ $(1 \leq i \leq L)$ and input constraint $x \in \mathcal{C}$. We have*

$$\min_{x \in \mathcal{C}} f(x) \geq \min_{x \in \mathcal{C}} \boldsymbol{a}_{\text{CROWN}}^\top x + c_{\text{CROWN}}$$

where $\boldsymbol{a}_{\text{CROWN}}$ and c_{CROWN} can be computed using $\mathbf{W}^{(i)}, \mathbf{b}^{(i)}, \mathbf{l}^{(i)}, \mathbf{u}^{(i)}$ in polynomial time.

When $\mathcal{C}$ is an ℓ_p norm ball, minimization over the linear function can be easily solved using Hölder's inequality. The main benefit of CROWN is its efficiency: CROWN can be efficiently implemented on machine learning accelerators such as GPUs [42] and TPUs [45], and it can be a few magnitudes faster than an LP verifier which is hard to parallelize on GPUs. CROWN was generalized to general architectures [42, 30] while we only demonstrate it for feedforward ReLU networks for simplicity.

2.2 Branch and Bound and Neuron Split Constraints

Branch and bound (BaB) method is widely adopted in complete verifiers [7]: we divide the domain of the verification problem $\mathcal{C}$ into two subdomains $\mathcal{C}_1 = \{x \in \mathcal{C}, z_j^{(i)} \geq 0\}$ and $\mathcal{C}_2 = \{x \in \mathcal{C}, z_j^{(i)} < 0\}$ where $z_j^{(i)}$ is an unstable ReLU neuron in $\mathcal{C}$ but now becomes linear for each subdomain. Incomplete verifiers can then estimate the lower bound of each subdomain with relaxations. If the lower bound produced for subdomain $\mathcal{C}_i$ (denoted by $\underline{f}_{\mathcal{C}_i}$) is greater than 0, $\mathcal{C}_i$ is verified; otherwise, we further branch over domain $\mathcal{C}_i$ by splitting another unstable ReLU neuron. The process terminates when all subdomains are verified. The completeness is guaranteed when all unstable ReLU neurons are split.

LP verifier with neuron split constraints. A popular incomplete verifier used in BaB is LP verifier. Essentially, when we split the j-th ReLU in layer i, we can simply add $z_j^{(i)} \geq 0$ or $z_j^{(i)} < 0$ to Eq. 1 and get a linearly relaxed lower bound to each subdomain. We denote the $\mathcal{Z}^{+(i)}$ and $\mathcal{Z}^{-(i)}$ as the

set of neuron indices with positive and negative split constraints in layer i. We define the split constraints at layer i as $\mathcal{Z}^{(i)} := \{z^{(i)} \mid z^{(i)}_{j_1} \geq 0, z^{(i)}_{j_2} < 0, \forall j_1 \in \mathcal{Z}^{+(i)}, \forall j_2 \in \mathcal{Z}^{-(i)}\}$. We denote the vector of all pre-ReLU neurons as z, and we define a set $\mathcal{Z}$ to represent the split constraints on z: $\mathcal{Z} = \mathcal{Z}^{(1)} \cap \mathcal{Z}^{(2)} \cap \cdots \cap \mathcal{Z}^{(L-1)}$. For convenience, we also use the shorthand $\tilde{\mathcal{Z}}^{(i)} := \mathcal{Z}^{(1)} \cap \cdots \cap \mathcal{Z}^{(i)}$ and $\tilde{z}^{(i)} := \{z^{(1)}, z^{(2)}, \cdots, z^{(i)}\}$. LP verifiers can easily handle these neuron split constraints but are very expensive than bound propagation methods like CROWN and cannot be accelerated on GPUs.

Branching strategy. Branching strategies (selecting which ReLU neuron to split) are generally agnostic to the incomplete verifier used in BaB but do affect the overall BaB performance. BaBSR [6] is a widely used strategy in complete verifiers, which is based on an fast estimates on objective improvements after splitting each neuron. The neuron with highest estimated improvement is selected for branching. Recently, Filtered Smart Branching (FSB) [10] improves BaBSR by mimicking strong branching - it utilizes bound propagation methods to evaluate the best a few candidates proposed by BaBSR and chooses the one with largest improvement. Graph neural network (GNN) based branching was also proposed [22]. Our β-CROWN BaB is a general complete verification framework fit for any potential branching strategy, and we evaluate both BaBSR and FSB in experiments.

3 β-CROWN for Complete and Incomplete Verification

In this section, we first give intuitions on how β-CROWN handles neuron split constraints without costly LP solvers. Then we formally state the main theorem of β-CROWN from both primal and dual spaces, and discuss how to tighten the bounds using free parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$. Lastly, we propose β-CROWN BaB, a complete verifier that is also a strong incomplete verifier when stopped early.

3.1 β-CROWN: Linear Bound Propagation with Neuron Split Constraints

The NN verification problem under neuron split constraints can be written as an optimization:

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x). \tag{2}$$

Bound propagation methods like CROWN can give a relatively tight lower bound for $\min_{x \in \mathcal{C}} f(x)$ but they *cannot handle the neuron split constraints* $z \in \mathcal{Z}$. Before we present our main theorem, we first show the intuition on how to apply split constraints to the bound propagation process.

To encode the neuron splits, we first define diagonal matrix $\mathbf{S}^{(i)} \in \mathbb{R}^{d_i \times d_i}$ in Eq. 3 where $i \in [1, \cdots L-1], j \in [1, \cdots, d_i]$:

$$\mathbf{S}^{(i)}_{j,j} = -1(\text{if split } z^{(i)}_j \geq 0); \quad \mathbf{S}^{(i)}_{j,j} = +1(\text{if split } z^{(i)}_j < 0); \quad \mathbf{S}^{(i)}_{j,j} = 0(\text{if not split } z^{(i)}_j) \tag{3}$$

We start from last layer and derive linear bounds for each intermediate layer $z^{(i)}$ and $\hat{z}^{(i)}$ for both $x \in \mathcal{C}$ and $z \in \mathcal{Z}$. We also assume that pre-ReLU bounds $\mathbf{l}^{(i)} \leq z^{(i)} \leq \mathbf{u}^{(i)}$ for each layer i are available. We initially have:

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) = \min_{x \in \mathcal{C}, z \in \mathcal{Z}} \mathbf{W}^{(L)} \hat{z}^{(L-1)} + \mathbf{b}^{(L)}. \tag{4}$$

Since $\hat{z}^{(L-1)} = \text{ReLU}(z^{(L-1)})$, we can apply Lemma 2.1 to relax the ReLU neuron at layer $L-1$, and obtain a linear lower bound for $f(x)$ w.r.t. $z^{(L-1)}$:

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) \geq \min_{x \in \mathcal{C}, z \in \mathcal{Z}} \mathbf{W}^{(L)} \mathbf{D}^{(L-1)} z^{(L-1)} + \text{const}.$$

To enforce the split neurons at layer $L-1$, we use a Lagrange function with $\boldsymbol{\beta}^{(L-1)\top} \mathbf{S}^{(L-1)}$ multiplied on $z^{(L-1)}$:

$$\begin{aligned} \min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) &\geq \min_{\substack{x \in \mathcal{C} \\ \tilde{z}^{(L-2)} \in \tilde{\mathcal{Z}}^{(L-2)}}} \max_{\boldsymbol{\beta}^{(L-1)} \geq 0} \mathbf{W}^{(L)} \mathbf{D}^{(L-1)} z^{(L-1)} + \boldsymbol{\beta}^{(L-1)\top} \mathbf{S}^{(L-1)} z^{(L-1)} + \text{const} \\ &\geq \max_{\boldsymbol{\beta}^{(L-1)} \geq 0} \min_{\substack{x \in \mathcal{C} \\ \tilde{z}^{(L-2)} \in \tilde{\mathcal{Z}}^{(L-2)}}} \left(\mathbf{W}^{(L)} \mathbf{D}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\top} \mathbf{S}^{(L-1)} \right) z^{(L-1)} + \text{const} \end{aligned} \tag{5}$$

The first inequality is due to the definition of the Lagrange function: we remove the constraint $z^{(L-1)} \in \mathcal{Z}^{(L-1)}$ and use a multiplier to replace this constraint. The second inequality is due to weak duality. Due to the design of $\mathbf{S}^{(L-1)}$, neuron split $z^{(L-1)}_j \geq 0$ has a negative multiplier $-\boldsymbol{\beta}^{(L-1)}_j$

and split $z_j^{(L-1)} < 0$ has a positive multiplier $\boldsymbol{\beta}_j^{(L-1)}$. Any $\boldsymbol{\beta}^{(L-1)} \geq 0$ yields a lower bound for the constrained optimization. Then we substitute $z^{(L-1)}$ with $\mathbf{W}^{(L-1)}\hat{z}^{(L-2)} + \mathbf{b}^{(L-1)}$ for next layer:

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta}^{(L-1)}\geq 0} \min_{\substack{x\in\mathcal{C} \\ \tilde{z}^{(L-2)}\in\tilde{\mathcal{Z}}^{(L-2)}}} \left(\mathbf{W}^{(L)}\mathbf{D}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\top}\mathbf{S}^{(L-1)}\right)\mathbf{W}^{(L-1)}\hat{z}^{(L-2)} + \text{const} \quad (6)$$

We define a matrix $\mathbf{A}^{(i)}$ to represent the linear relationship between $f(x)$ and $\hat{z}^{(i)}$, where $\mathbf{A}^{(L-1)} = \mathbf{W}^{(L)}$ according to Eq. 4 and $\mathbf{A}^{(L-2)} = (\mathbf{A}^{(L-1)}\mathbf{D}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\top}\mathbf{S}^{(L-1)})\mathbf{W}^{(L-1)}$ by Eq. 6. Considering 1-dimension output $f(x)$, $\mathbf{A}^{(i)}$ has only 1 row. With $\mathbf{A}^{(L-2)}$, Eq. 6 becomes:

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta}^{(L-1)}\geq 0} \min_{\substack{x\in\mathcal{C} \\ \tilde{z}^{(L-2)}\in\tilde{\mathcal{Z}}^{(L-2)}}} \mathbf{A}^{(L-2)}\hat{z}^{(L-2)} + \text{const},$$

which is in a form similar to Eq. 4 except for the outer maximization over $\boldsymbol{\beta}^{(L-1)}$. This allows the back-substitution process (Eq. 4, Eq. 5, and Eq. 6) to continue. In each step, we swap $\max$ and $\min$ as in Eq. 5, so every maximization over $\boldsymbol{\beta}^{(i)}$ is outside of $\min_{x\in\mathcal{C}}$. Eventually, we have:

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta}\geq 0} \min_{x\in\mathcal{C}} \mathbf{A}^{(0)}x + \text{const},$$

where $\boldsymbol{\beta} := \left[\boldsymbol{\beta}^{(1)\top}\ \boldsymbol{\beta}^{(2)\top}\ \cdots\ \boldsymbol{\beta}^{(L-1)\top}\right]^\top$ concatenates all $\boldsymbol{\beta}^{(i)}$ vectors. Following the above idea, we present the main theorem in Theorem 3.1 (proof is given in Appendix A).

Theorem 3.1 (β-CROWN bound)**.** *Given an L-layer NN $f(x) : \mathbb{R}^{d_0} \to \mathbb{R}$ with weights $\mathbf{W}^{(i)}$, biases $\mathbf{b}^{(i)}$, pre-ReLU bounds $\mathbf{l}^{(i)} \leq z^{(i)} \leq \mathbf{u}^{(i)}$ ($1 \leq i \leq L$), input bounds $\mathcal{C}$, split constraints $\mathcal{Z}$. We have:*

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta}\geq 0} \min_{x\in\mathcal{C}} (\boldsymbol{a} + \mathbf{P}\boldsymbol{\beta})^\top x + \mathbf{q}^\top\boldsymbol{\beta} + c, \quad (7)$$

where $\boldsymbol{a} \in \mathbb{R}^{d_0}$,$\mathbf{P} \in \mathbb{R}^{d_0\times(\sum_{i=1}^{L-1} d_i)}$,$\mathbf{q} \in \mathbb{R}^{\sum_{i=1}^{L-1} d_i}$ and $c \in \mathbb{R}$ are functions of $\mathbf{W}^{(i)}$, $\mathbf{b}^{(i)}$, $\mathbf{l}^{(i)}$, $\mathbf{u}^{(i)}$.

Detailed formulations for $\boldsymbol{a}$, $\mathbf{P}$, $\mathbf{q}$ and c are given in Appendix A. Theorem 3.1 shows that when neuron split constraints exist, $f(x)$ can still be bounded by a linear equation containing optimizable multipliers $\boldsymbol{\beta}$. Observing Eq. 5, the main difference between CROWN and β-CROWN lies in the relaxation of each ReLU layer, where we need an extra term $\boldsymbol{\beta}^{(i)\top}\mathbf{S}^{(i)}$ in the linear relationship matrix (for example, $\mathbf{W}^{(L)}\mathbf{D}^{(L-1)}$ in Eq. 5) between $f(x)$ and $z^{(i)}$ to enforce neuron split constraints. This extra term in every ReLU layer yields $\mathbf{P}$ and $\mathbf{q}$ in Eq. 7 after bound propagations.

To solve the optimization problem in Eq. 7, we note that in the ℓ_p norm robustness setting ($\mathcal{C} = \{x \mid \|x - x_0\|_p \leq \epsilon\}$), the inner minimization has a closed solution:

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta}\geq 0} -\|\boldsymbol{a} + \mathbf{P}\boldsymbol{\beta}\|_q\epsilon + (\mathbf{P}^\top x_0 + \mathbf{q})^\top\boldsymbol{\beta} + \boldsymbol{a}^\top x_0 + c := \max_{\boldsymbol{\beta}\geq 0} g(\boldsymbol{\beta}) \quad (8)$$

where $\frac{1}{p} + \frac{1}{q} = 1$. The maximization is concave in $\boldsymbol{\beta}$ ($q \geq 1$), so we can simply optimize it using projected (super)gradient ascent with gradients from an automatic differentiation library. Since any $\boldsymbol{\beta} \geq 0$ yields a valid lower bound for $\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x)$, convergence is not necessary to guarantee soundness. β-CROWN is efficient - it has the same asymptotic complexity as CROWN when $\boldsymbol{\beta}$ is fixed. When $\boldsymbol{\beta} = 0$, β-CROWN yields the same results as CROWN; however the additional optimizable $\boldsymbol{\beta}$ allows us to maximize and tighten the lower bound due to neuron split constraints.

We define $\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^{d_i}$ for free variables associated with unstable ReLU neurons in Lemma 2.1 for layer i and define all free variables $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}^{(1)} \cdots \boldsymbol{\alpha}^{(L-1)}\}$. Since any $0 \leq \boldsymbol{\alpha}_j^{(i)} \leq 1$ yields a valid bound, we can optimize it to tighten the bound. Formally, we rewrite Eq. 8 with $\boldsymbol{\alpha}$ explicitly:

$$\min_{x\in\mathcal{C}, z\in\mathcal{Z}} f(x) \geq \max_{0\leq\boldsymbol{\alpha}\leq 1,\ \boldsymbol{\beta}\geq 0} g(\boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (9)$$

3.2 Connections to the Dual Problem

In this subsection, we show that β-CROWN can also be derived from a dual LP problem. Based on Eq. 1 and linear relaxations in Section 2, we first construct an LP problem for ℓ_∞ robustness verification in Eq. 10 where $i \in \{1, \cdots, L-1\}$.

$$
\begin{aligned}
&\min\ f(x) := z^{(L)}(x) \quad \text{s.t.} \\
&\text{Network and Input Bounds: } z^{(i)} = \mathbf{W}^{(i)}\hat{z}^{(i-1)} + \mathbf{b}^{(i)}; \hat{z}^{(0)} \geq x_0 - \epsilon; \hat{z}^{(0)} \leq x_0 + \epsilon; \\
&\text{Stable: } \hat{z}^{(i)} = z^{(i)} \text{ (if } \mathbf{l}_j^{(i)} \geq 0); \hat{z}^{(i)} = 0 \text{ (if } \mathbf{u}_j^{(i)} \leq 0); \\
&\text{Unstable: } \hat{z}_j^{(i)} \geq 0, \hat{z}_j^{(i)} \geq z_j^{(i)}, \hat{z}_j^{(i)} \leq \frac{\mathbf{u}_j^{(i)}}{\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}} \left(z_j^{(i)} - \mathbf{l}_j^{(i)}\right) \text{ (if } \mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(u)}, j \notin \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)}) \\
&\text{Neuron Split Constraints: } \hat{z}_j^{(i)} = z_j^{(i)}, z_j^{(i)} \geq 0 \text{ (if } j \in \mathcal{Z}^{+(i)}); \hat{z}_j^{(i)} = 0, z_j^{(i)} < 0 \text{ (if } j \in \mathcal{Z}^{-(i)})
\end{aligned}
\tag{10}
$$

Compared to the formulation in [40], we have neuron split constraints. Many BaB based complete verifiers [7, 22] use an LP solver for Eq. 10 as the incomplete verifier. We first show that it is possible to derive Theorem 3.1 from the dual of this LP, leading to Theorem 3.2:

Theorem 3.2. *The objective d_{LP} for the dual problem of Eq. 10 can be represented as*

$$
d_{LP} = -\|\boldsymbol{a} + \mathbf{P}\boldsymbol{\beta}\|_1 \cdot \epsilon + (\mathbf{P}^\top x_0 + \mathbf{q})^\top \boldsymbol{\beta} + \boldsymbol{a}^\top x_0 + c,
$$

where $\boldsymbol{a}$, $\mathbf{P}$, $\mathbf{q}$ and c are defined in the same way as in Theorem 3.1, and $\boldsymbol{\beta} \geq 0$ corresponds to the dual variables of neuron split constraints in Eq. 10.

A similar connection between CROWN and dual LP based verifier [40] was shown in [29], and their results can be seen as a special case of ours when $\boldsymbol{\beta} = 0$ (none of the split constraints are active). An immediate consequence is that β-CROWN can potentially solve Eq. 10 as well as using an LP solver:

Corollary 3.2.1. *When $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are optimally set and intermediate bounds $\mathbf{l}, \mathbf{u}$ are fixed, β-CROWN produces p^*_{LP}, the optimal objective of LP with split constraints in Eq. 10:*

$$
\max_{0 \leq \boldsymbol{\alpha} \leq 1, \boldsymbol{\beta} \geq 0} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = p^*_{LP},
$$

In Appendix A, we give detailed formulations for conversions between the variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ in β-CROWN and their corresponding dual variables in the LP problem.

3.3 Joint Optimization of Free Variables in β-CROWN

In Eq. 9, g is also a function of $\mathbf{l}_j^{(i)}$ and $\mathbf{u}_j^{(i)}$, the intermediate layer bounds for each neuron $z_j^{(i)}$. They are also computed using β-CROWN. To obtain $\mathbf{l}_j^{(i)}$, we set $f(x) := z_j^{(i)}(x)$ and apply Theorem 3.1:

$$
\min_{x \in \mathcal{C}, \tilde{z}^{(i-1)} \in \tilde{\mathcal{Z}}^{(i-1)}} z_j^{(i)}(x) \geq \max_{0 \leq \boldsymbol{\alpha}' \leq 1,\ \boldsymbol{\beta}' \geq 0} g'(\boldsymbol{\alpha}', \boldsymbol{\beta}') := \mathbf{l}_j^{(i)} \tag{11}
$$

and for $\mathbf{u}_j^{(i)}$ we set $f(x) := -z_j^{(i)}(x)$. Importantly, during solving these intermediate layer bounds, the $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ are *independent sets of variables*, not the same ones for the objective $f(x) := z^{(L)}$. Since g is a function of $\mathbf{l}_j^{(i)}$, it is also a function of $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$. In fact, there are a total of $\sum_{i=1}^{L-1} d_i$ intermediate layer neurons, and each neuron is associated with a set of independent $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ variables. Optimizing these variables allowing us to tighten the relaxations on unstable ReLU neurons (which depend on $\mathbf{l}_j^{(i)}$ and $\mathbf{u}_j^{(i)}$) and produce tight final bounds, which is impossible in LP. In other words, we need to optimize $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$, which are two vectors concatenating $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ as well as a large number of $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ used to compute each intermediate layer bound:

$$
\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) \geq \max_{0 \leq \hat{\boldsymbol{\alpha}} \leq 1,\ \hat{\boldsymbol{\beta}} \geq 0} g(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}). \tag{12}
$$

This formulation is non-convex and has a large number of variables. Since any $0 \leq \hat{\boldsymbol{\alpha}} \leq 1, \hat{\boldsymbol{\beta}} \geq 0$ leads to a valid lower bound, the non-convexity does not affect soundness. When intermediate layer bounds are also allowed to be tightened during optimization, we can outperform the LP verifier for Eq. 10 using fixed intermediate layer bounds. Typically, when Eq. 10 is formed, intermediate layer bounds are pre-computed with bound propagation procedures [7, 22], which are far from optimal.

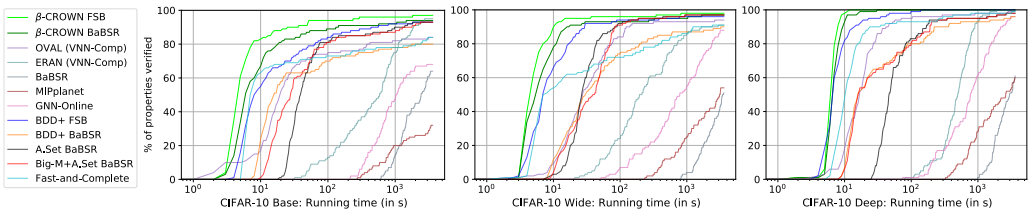
To estimate the dimension of this problem, we denote the number of unstable neurons at layer i as $s_i := \text{Tr}(|\mathbf{S}^{(i)}|)$. Each neuron in layer i is associated with $2 \times \sum_{k=1}^{i-1} s_k$ variables $\boldsymbol{\alpha}'$. Suppose each hidden layer has d neurons ($s_i = O(d)$), then $\hat{\boldsymbol{\alpha}}$ has $2 \times \sum_{i=1}^{L-1} d_i \sum_{k=1}^{i-1} s_k = O(L^2 d^2)$ variables in total. This can be too large for efficient optimization, so we share $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ among the intermediate neurons of the same layer, leading to a total number of $O(L^2 d)$ variables to optimize. Note that a weaker form of joint optimization was also discussed in [43] without $\boldsymbol{\beta}$, and a detailed analysis can be found in Appendix B.2.

Table 1: Average runtime and average number of branches on three CIFAR-10 models over 100 properties.

Method	CIFAR-10 Base			CIFAR-10 Wide			CIFAR-10 Deep		
	time(s)	branches	%timeout	time(s)	branches	%timeout	time(s)	branches	%timeout
BaBSR [6]	2367.78	1020.55	36.00	2871.14	812.65	49.00	2750.75	401.28	39.00
MIPplanet [14]	2849.69	-	68.00	2417.53	-	46.00	2302.25	-	40.00
ERAN* [33, 31, 34, 32]	805.94	-	5.00	632.20	-	9.00	545.72	-	**0.00**
GNN-online [22]	1794.85	565.13	33.00	1367.38	372.74	15.00	1055.33	131.85	4.00
BDD+ BaBSR [5]	807.91	195480.14	20.00	505.65	74203.11	10.00	266.28	12722.74	4.00
OVAL (BDD+ GNN)* [5, 22]	662.17	67938.38	16.00	280.38	17895.94	6.00	94.69	1990.34	1.00
A.set BaBSR [9]	381.78	12004.60	7.00	165.91	2233.10	3.00	190.28	2491.55	2.00
BigM+A.set BaBSR [9]	390.44	11938.75	7.00	172.65	4050.59	3.00	177.22	3275.25	2.00
Fast-and-Complete [43]	695.01	119522.65	17.00	495.88	80519.85	9.00	105.64	2455.11	1.00
BaDNB (BDD+ FSB)* [10]	309.29	38239.04	7.00	165.53	11214.44	4.00	10.50	368.16	**0.00**
β-CROWN BaBSR (ours)	266.05	142235.50	6.00	152.70	65708.48	3.00	8.71	314.66	**0.00**
β-CROWN FSB (ours)	**145.68**	60867.21	**3.00**	**96.45**	35721.14	**2.00**	**7.56**	49.34	**0.00**

* OVAL (BDD+ GNN) and ERAN are VNN competition 2020 winners; BaDNB (BDD+ FSB) is a concurrent work.

Figure 1: Percentage of solved properties with growing running time. β-CROWN FSB (light green) and β-CROWN BaBSR (dark green) clearly lead in all 3 settings and solve over 80% properties within 10 seconds.



3.4 β-CROWN with Branch and Bound (β-CROWN BaB)

We perform complete verification following BaB framework [7] using β-CROWN as the incomplete solver, and we use simple branching heuristics like BaBSR [6] or FSB [10]. To efficiently utilize GPU, we also use batch splits to evaluate multiple subdomains in the same batch as in [42, 9]. We list our full algorithm β-CROWN BaB in Appendix B and we show it is sound and complete here:

Theorem 3.3. *β-CROWN with Branch and Bound on splitting ReLUs is sound and complete.*

Soundness is trivial because β-CROWN is a sound verifier. For completeness, it suffices to show that when all unstable ReLU neurons are split, β-CROWN gives the global minimum for Eq. 10. In contrast, combining CROWN [44] with BaB does *not* yield a complete verifier, as it cannot detect infeasible splits and a slow LP solver is still needed to guarantee completeness [43]. Instead, β-CROWN can detect infeasible subdomains - according to duality theory, an infeasible primal problem leads to an unbounded dual objective, which can be detected (see Sec. B.3 for more details).

Additionally, we show the potential of *early stopping a complete verifier as an incomplete verifier*. BaB approaches the exact solution of Eq. 1 by splitting the problem into multiple subdomains, and more subdomains give a tighter lower bound for Eq. 1. Unlike traditional complete verifiers, β-CROWN is efficient to explore a large number of subdomains during a very short time, making β-CROWN BaB an attractive solution for efficient incomplete verification.

4 Experimental Results

4.1 Comparison to Complete Verifiers

We evaluate our performance on complete verification on dataset provided in [22, 9] and used in VNN competition 2020 [21]. The dataset contains three CIFAR-10 models (Base, Wide, and Deep) with 100 examples each. Each data example is associated with an ℓ_∞ norm ϵ and a target label for verification (referred to as a *property* to verify). The details of neural network structures and experimental setups can be found in Appendix C. We compare against multiple baselines for complete verification: (1) BaBSR [6], a basic BaB and LP based verifier; (2) MIPplanet [14], a customized MIP solver for NN verification where unstable ReLU neurons are randomly selected for splitting; (3) ERAN [33, 31, 34, 32], an abstract interpretation based verifier which is one of VNN competition 2020 winners with lowest timeout rate; (4) GNN-Online [22], a BaB and LP based verifiers using a learned Graph Neural Network (GNN) to guide the ReLU splits; (5) BDD+ BaBSR [5], a verification framework based on Lagrangian decomposition on GPUs (BDD+) with BaBSR branching strategy; (6) OVAL (BDD+ GNN) [5, 22], one of VNN competition 2020 winners, using BDD+ with GNN guiding the ReLU splits; (7) A.set BaBSR and (8) Big-M+A.set BaBSR [9], very recent dual-space

Table 2: **Verified accuracy (%)** and avg. time (s) of 1000 images evaluated on the ERAN models in [33, 35, 26].

Dataset (Same settings as [33, 35, 26])	Model	CROWN/DeepPoly* [34] Verified%	Time (s)	kPoly [33] Ver.%	Time(s)	OptC2V [35] Ver.%	Time(s)	PRIMA† [26] Ver.%	Time(s)	β-CROWN FSB Ver.%	Time(s)	Upper bound
MNIST	ConvSmall	15.8	3	34.7	477	43.6	55	59.8	42	**71.6**	46	74.6
	ConvBig	71.1	21	73.6	40	77.1	102	77.5	15	**77.7**	78	80.4
CIFAR	ConvSmall	35.9	4	39.9	86	39.8	105	44.1	20	**46.3**	18	48.2
	ConvBig	42.1	43	45.9	346	No public code		46.9	97	**50.3**	55	61.3
	ResNet	24.3	12	24.5	91	cannot run		24.9	64	**25.1**	42	29.0

* CROWN/DeepPoly evaluated on CPU. † PRIMA is a concurrent work and results are directly from [26] under the same setting.

Table 3: **Verified accuracy (%)** and avg. per-example verification time (s) on 7 models from SDP-FO [8]. CROWN/DeepPoly are fast but loose bound propagation based methods, and they cannot be improved with more running time. SDP-FO uses stronger semidefinite relaxations, which can be very slow and sometimes has convergence issues. PRIMA is one of the best convex relaxation barrier breaking methods; we did not include kPoly and OptC2V because they are weaker than PRIMA (see Table 2).

Dataset $\epsilon = 0.3$ and $\epsilon = 2/255$	Model	CROWN/DeepPoly Verified%	Time (s)	SDP-FO [8]* Ver.%	Time(s)	PRIMA [26] Ver.%	Time(s)	β-CROWN FSB Ver.%	Time(s)	Upper bound
MNIST	CNN-A-Adv	1.0	1	43.4	>20h	44.5	136	**68.0**	76	79.5
CIFAR	CNN-B-Adv	21.5	2	32.8	>25h	38.0	360	**44.5**	94	64.0
	CNN-B-Adv-4	43.5	2	46.0	>25h	53.5	51	**54.0**	52	62.5
	CNN-A-Adv	35.5	2	39.6	>25h	41.5	11	**43.5**	31	52.0
	CNN-A-Adv-4	41.5	1	40.0	>25h	45.0	7	**46.0**	4	49.5
	CNN-A-Mix	23.5	1	39.6	>25h	37.5	36	**41.5**	33	51.5
	CNN-A-Mix-4	38.0	1	47.8	>25h	48.5	9	**50.5**	8	55.0

* SDP-FO results are directly from their paper due to very long running time. † PRIMA and β-CROWN FSB results are on the same set of 200 random examples. β-CROWN uses **1** GPU and **1** CPU; PRIMA uses **1** GPU and **20** CPUs.

verifiers on GPUs with a tighter linear relaxation than LP; (9) Fast-and-Complete [43], which uses CROWN on GPUs as the incomplete verifier in BaB without neuron split constraints; (10) BaDNB (BDD+ FSB) [10], a concurrent state-of-the-art complete verifier, using BDD+ on GPUs with FSB branching strategy. β-CROWN BaB can use either BaBSR or FSB branching heuristic, and we include both in evaluation. All methods use 1 CPU + 1 GPU with a 1 hour timeout threshold.

We report the average verification time and branch numbers in Table 1 and plot the percentage of solved properties over time in Figure 1. β-CROWN FSB achieves the fastest average running time compared to all other baselines with minimal timeouts, and also clearly leads on the cactus plot. When using a weaker branching heuristic, β-CROWN BaBSR still outperforms all literature baselines, including very recent ones such as A.set BaBSR [9], Fast-and-Complete [43] and BaDNB [10]. Our benefits are more clearly shown in Figure 1, where we solve 80% to 90% examples under 10 seconds and most other verifiers can only verify a small portion or none of the properties within 10 seconds.

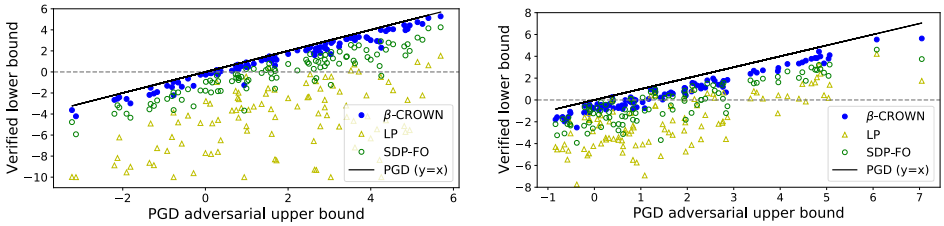
4.2 Comparison to Incomplete Verifiers

Verified accuracy. In Table 2, we compare against a few representative and strong incomplete verifiers on 5 convolutional networks for MNIST and CIFAR-10 under the same set of 1000 images and perturbation ϵ as reported in [35, 35, 26]. Among the baselines, kPoly [33], OptC2V [35] and PRIMA [26] utilize state-of-the-art multi-neuron linear relaxation for ReLUs and can bypass the single-neuron convex relaxation barrier [29], and are among the strongest incomplete verifiers. β-CROWN FSB achieves better verified accuracy on all five models using a similar amount of time. Some models, such as MNIST ConvBig and CIFAR ResNet, are quite challenging - the verified accuracy obtained by β-CROWN FSB is close to the upper bound found via PGD attack.

To make more comprehensive evaluations, in Table 3 we further compare against a state-of-the-art semidefinite programming (SDP) based verifier, SDP-FO [8], on one MNIST and six CIFAR-10 models reported in their paper. The models were trained using adversarial training, which posed a challenge for verification [27]. The SDP formulation is usually tighter than linear relaxation based ones, but it is computationally expensive - SDP-FO takes 2 to 3 hours to converge on one GPU for verifying a single property, resulting 5,400 GPU hours to verify 200 testing images with all 9 labels. Due to resource limitations, we directly quote SDP-FO results from [8] on the same set of models. We evaluate verified accuracy on the same set of 200 test images for other baselines. We include PRIMA [25], the strongest baseline reported in Table 2, as a baseline, which generally outperforms kPoly and OptC2V. Table 3 shows that overall we are three orders of magnitude faster than SDP-FO while still achieving consistently higher (8% in average) verified accuracy on average.

Tightness of verification. In Figure 2, we compare the tightness of verification bounds against SDP-FO on two adversarially trained networks from [8]. Specifically, we use the verification objective

Figure 2: Verified lower bound v.s. PGD adversarial upper bound. A lower bound closer to the upper bound (closer to the line $y = x$) is better. β-CROWN FSB uses 3mins while SDP-FO needs roughly 3 hours per point.



(a) MNIST CNN-A-Adv, runner-up targets, $\epsilon = 0.3$ (b) CIFAR CNN-B-Adv, runner-up targets, $\epsilon = 2/255$

$f(x) := z_y^{(L)}(x) - z_{y'}^{(L)}(x)$, where $z^{(L)}$ is the logit layer output, y and y' are the true label and the runner-up label. For each test image, PGD attack [23] can provide an adversarial upper bound $\overline{f}$ of the optimal objective: $f^* \leq \overline{f}$. Verifiers, on the other hand, can provide a verified lower bound $\underline{f} \leq f^*$. Bounds from tighter verification methods lie closer to line $y = x$ in Figure 2. Figure 2 shows that on both PGD adversarially trained networks, β-CROWN FSB consistently outperforms SDP-FO for all 100 random test images. Importantly, for each point on the plots, β-CROWN FSB needs only 3 minutes while SDP-FO needs 178 minutes on average. LP verifier produces much looser bounds than β-CROWN FSB and SDP-FO. Additional results are in Appendix C.2.

5 Related Work

Early complete verifiers relied on existing solvers [19, 14, 18, 11, 36] and were limited to very small problem instances. BaB based method was proposed to better exploit the network structure using LP-based incomplete verifier for bounding and ReLU splits for branching [7, 38, 22, 4]. Input domain branching was also considered in [39, 28, 1] but limited by input dimensions [7]. Most traditional complete verifiers need hours to verify one example on networks with a few thousand neurons.

Recently, instead of using LP for bounding, a few approaches have been proposed to use efficient iterative solvers or bound propagation methods on GPUs. Bunel et al. [5] decomposed the verification problem layer by layer, solved each layer in a closed form on GPUs, and used Lagrangian to enforce consistency between layers. However, their formulation only has the same power as LP and needs many iterations to converge. De Palma et al. [9] used a dual-space verifier with a tighter linear relaxation [2, 35] than LP at a cost of high complexity and potentially large number of constraints. A concurrent work BaDNB [10] proposed a new branching strategy, filtered smart branching (FSB), combined with Lagrangian decomposition to get better verification performance. Xu et al. [42] used CROWN as a massively paralleled incomplete solver on GPUs for complete verification, but it cannot handle neuron split constraints, leading to suboptimal efficiency and high timeout rates.

For incomplete verification, Salman et al. [29] shows the inherent limitation of using per-neuron convex relaxations for verification problems. Singh et al. [33] and Müller et al. [26] broke this barrier by relaxing multiple ReLU neurons; Tjandraatmadja et al. [35] considered relaxing a layer of neurons using a strong mixed-integer programming formulation [1]. SDP based relaxations [27, 15, 13, 8] typically produce tight bounds but with significantly higher cost. In this work, we impose neuron split constraints using β-CROWN, break this barrier using branch and bound, and outperform existing incomplete verifiers under many scenarios in both runtime and tightness.

Additionally, another line of works train networks to enhance verified accuracy, typically using cheap incomplete verifiers at training time [40, 37, 24, 41, 17, 24, 45, 3]. Traditionally only these verification-customized networks can have reasonable verified accuracy, while β-CROWN BaB can also give non-trivial verified accuracy on relatively large networks agnostic to verification.

6 Conclusion

We proposed β-CROWN, a new bound propagation method that can fully encode the neuron split constraints introduced in BaB, which clearly leads in both complete and incomplete verification settings. The success of β-CROWN comes from a few factors: (1) In Section 3.1, we show that β-CROWN is an GPU-friendly bound propagation algorithm *significantly faster than LP solvers*. (2) In Section 3.2, we show that β-CROWN is solving an equivalent problem of the LP verifier *with neuron split constraints*. (3) In Section 3.3, we show that β-CROWN can jointly optimize intermediate layer bounds and *achieve tighter bounds than typical LP verifiers* using fixed intermediate layer bounds.

References

[1] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019.

[2] R. Anderson, J. Huchette, C. Tjandraatmadja, and J. P. Vielma. Strong convex relaxations and mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

[3] M. Balunovic and M. Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations (ICLR)*, 2020.

[4] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[5] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. H. S. Torr, and M. P. Kumar. Lagrangian decomposition for neural network verification. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

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

[7] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[8] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[9] A. De Palma, H. S. Behl, R. Bunel, P. H. S. Torr, and M. P. Kumar. Scaling the convex barrier with active sets. *International Conference on Learning Representations (ICLR)*, 2021.

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

[11] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, 2018.

[12] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[13] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli. Efficient neural network verification with exactness characterization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

[14] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

[15] M. Fazlyab, M. Morari, and G. J. Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 2020.

[16] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.

[17] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[18] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

[19] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[21] C. Liu and T. Johnson. Vnn comp 2020. URL https://sites.google.com/view/vnn20/vnncomp.

[22] J. Lu and M. P. Kumar. Neural network branching for neural network verification. *International Conference on Learning Representation (ICLR)*, 2020.

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

[24] M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

[25] C. Müller, G. Singh, M. Püschel, and M. Vechev. Neural network robustness verification on gpus. *arXiv preprint arXiv:2007.10868*, 2020.

[26] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021.

[27] A. Raghunathan, J. Steinhardt, and P. S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[28] V. R. Royo, R. Calandra, D. M. Stipanovic, and C. Tomlin. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019.

[29] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[30] Z. Shi, H. Zhang, K.-W. Chang, M. Huang, and C.-J. Hsieh. Robustness verification for transformers. In *International Conference on Learning Representations (ICLR)*, 2020.

[31] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[32] G. Singh, T. Gehr, M. Püschel, and M. Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018.

[33] G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

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

[35] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. Patel, and J. P. Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[36] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *International Conference on Learning Representations (ICLR)*, 2019.

[37] S. Wang, Y. Chen, A. Abdou, and S. Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[38] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[39] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018.

[40] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.

[41] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[42] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[43] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *International Conference on Learning Representations (ICLR)*, 2021.

[44] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[45] H. Zhang, H. Chen, C. Xiao, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 2.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs can be found in supplementary
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] The verification results are deterministic.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We have cited all the creators of the benchmark datasets in Section 4.
   (b) Did you mention the license of the assets? [No] The benchmarks we tested are under MIT license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]