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

Shiqi Wang[*,1] **Huan Zhang**[*,2] **Kaidi Xu**[*,3]
Xue Lin[3] **Suman Jana**[1] **Cho-Jui Hsieh**[4] **Zico Kolter**[2]
[1]Columbia University [2]CMU [3]Northeastern University [4]UCLA
sw3215@columbia.edu huan@huan-zhang.com xu.kaid@northeastern.edu
xue.lin@northeastern.edu suman@cs.columbia.edu chohsieh@cs.ucla.edu
zkolter@cs.cmu.edu
* *Equal Contribution*

Abstract

Bound propagation based incomplete neural network verifiers such as CROWN are very efficient and can significantly accelerate branch-and-bound (BaB) based complete verification of neural networks. However, bound propagation cannot fully handle the neuron split constraints introduced by BaB commonly handled by expensive linear programming (LP) solvers, leading to loose 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 up to three orders of magnitude faster than LP-based BaB methods, and is notably faster than all existing approaches while producing lower timeout rates. By terminating BaB early, our method can also be used for efficient incomplete verification. We consistently achieve higher verified accuracy in many settings compared to 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. Our algorithm empowered the α,β-CROWN (`alpha-beta-CROWN`) verifier, the winning tool in VNN-COMP 2021. Our code is available at `http://PaperCode.cc/BetaCROWN`.

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.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

We first focus on *complete* verification: the verifier should give a definite "yes/no" answer given sufficient time. Many complete verifiers rely on the branch and bound (BaB) method [8] involving (1) branching by recursively splitting the original verification problem into subdomains (e.g., splitting a ReLU neuron into positive/negative linear regions by adding split constraints) and (2) bounding each subdomain with specialized incomplete verifiers. Traditional BaB-based verifiers use expensive linear programming (LP) solvers [15, 23, 7] as incomplete verifiers which can fully encode neuron split constraints. Meanwhile, a recent verifier, Fast-and-Complete [45], demonstrates that cheap incomplete verifiers can significantly accelerate complete verification on GPUs over LP-based ones thanks to their efficiency. Many cheap incomplete verifiers are based on *bound propagation methods* [46, 42, 41, 13, 17, 36, 44], i.e., maintaining and propagating tractable and sound bounds through networks, and CROWN [46] is a representative which propagates a linear or quadratic bound.

However, unlike LP based verifiers, existing bound propagation methods lack the power to handle neuron split constraints introduced by BaB. For instance, given inputs $x, y \in [-1, 1]$, they can bound a ReLU's input $x + y$ as $[-2, 2]$ but they have no means to consider neuron split constraints such as $x - y \geq 0$ introduced by splitting another ReLU to the positive linear region. Such a problem causes looser bounds and unnecessary branching, hurting the verification efficiency. Even worse, without considering these split constraints, bound propagation methods cannot detect many infeasible subdomains in BaB [45], leading to incompleteness unless costly checking is performed.

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 $\boldsymbol{\beta}$ which come from propagation of Lagrangian multipliers, 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 intermediate layer bounds similar to [44] but also with the additional parameters $\boldsymbol{\beta}$, 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 [8] by up to *three orders of magnitudes* on a commonly used benchmark suite on CIFAR-10 [6, 10]. Compared to all state-of-the-art GPU-based complete verifiers [7, 45, 10, 23, 6, 11], our approach is noticeably faster with lower timeout rates. Our algorithm empowered the tool α,β-CROWN (`alpha-beta-CROWN`), which won the 2nd International Verification of Neural Networks Competition [3] (VNN-COMP 2021) with the highest total score and verified the most number of problem instances in 8 benchmarks.

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 tight linear relaxations [35, 37, 26] and semidefinite relaxations [9]. Compared to the typically tightest but very costly incomplete verifier SDP-FO [9] based on the semidefinite programming (SDP) relaxations [28, 14], our method obtains consistently higher verified accuracy while reducing verification time by three orders of magnitudes.

2 Background

2.1 The neural network verification problem and its LP relaxation

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 $\underline{f} \leq f^*$. If $\underline{f} \geq 0$, then $f^* > 0$ so $f(x)$ can be verified; when $\underline{f} < 0$, we are not able to infer the sign of f^* so cannot conclude if the specification holds or not.

A commonly used incomplete verification technique is to relax non-convex ReLU constraints 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 the "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: (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 [15, 42]. 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.

2.2 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 [46] 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 $w, v \in \mathbb{R}^d, \mathbf{l} \leq v \leq \mathbf{u}$ (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 [46])**.** *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*

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

where $\boldsymbol{a}_{\mathrm{CROWN}}$ 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 [44] and TPUs [47], 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 [44, 31] while we only demonstrate it for feedforward ReLU networks for simplicity. Additionally, Xu et al. [45] showed that it is possible to optimize the slope of the lower bound, $\boldsymbol{\alpha}$, using gradient ascent, to further tighten the bound (sometimes referred to as α-CROWN).

2.3 Branch and Bound and Neuron Split Constraints

Branch and bound (BaB) method is widely adopted in complete verifiers [8]: 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 the 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 $\mathcal{Z}^{(i)} := \{z^{(i)} \mid z_{j_1}^{(i)} \geq 0, z_{j_2}^{(i)} < 0, \forall j_1 \in \mathcal{Z}^{+(i)}, \forall j_2 \in \mathcal{Z}^{-(i)}\}$. 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 more 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 [7] 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) [11] 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 [23]. 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 seen as an optimization problem:

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

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 $i \in [1, \cdots L-1], j \in [1, \cdots, d_i]$ are indices of layers and neurons, respectively:

$$\mathbf{S}_{j,j}^{(i)} = -1(\text{if split } z_j^{(i)} \geq 0); \quad \mathbf{S}_{j,j}^{(i)} = +1(\text{if split } z_j^{(i)} < 0); \quad \mathbf{S}_{j,j}^{(i)} = 0(\text{if no split } z_j^{(i)}) \tag{3}$$

We start from the last layer and derive linear bounds for each intermediate layer $z^{(i)}$ and $\hat{z}^{(i)}$ with both constraints $x \in \mathcal{C}$ 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 (see discussions in Sec. 3.3 on these intermediate layer bounds). 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 $\hat{z}^{(L-1)} = \mathrm{ReLU}(z^{(L-1)})$, 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. $z^{(L-1)}$ (we omit all constant terms to avoid clutter):

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

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 $z_j^{(L-1)} \geq 0$ has a negative multiplier $-\boldsymbol{\beta}_j^{(L-1)}$ 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 problem. Then we substitute $z^{(L-1)}$ with $\mathbf{W}^{(L-1)}\hat{z}^{(L-2)} + \mathbf{b}^{(L-1)}$ for next layer:

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

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:*

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

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:

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

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, similarly as done in [45]. Formally, we rewrite Eq. 8 with $\boldsymbol{\alpha}$ explicitly:

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

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 $i \in \{1, \cdots, L-1\}$.

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

Compared to the formulation in [42], we have neuron split constraints. Many BaB based complete verifiers [8, 23] 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 [42] was shown in [30], 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:

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

For computing $\mathbf{u}_j^{(i)}$ we simply 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, in many previous works [8, 23, 6], when the LP formulation Eq. 10 is formed, intermediate layer bounds are pre-computed with bound propagation procedures [8, 23], 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 $2 \times \sum_{i=1}^{L-1} d_i \sum_{k=1}^{i-1} s_k = O(L^2d^2)$ 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 $O(L^2d)$ variables to optimize. Note that a weaker form of joint optimization was also discussed in [45] without $\boldsymbol{\beta}$, and a detailed analysis can be found in Appendix B.2.

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

We perform complete verification following BaB framework [8] using β-CROWN as the incomplete solver, and we use simple branching heuristics like BaBSR [7] or FSB [11]. To efficiently utilize GPU, we also use batch splits to evaluate multiple subdomains in the same batch as in [44, 10]. 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 [46] 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 [45]. 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 complete verification performance on dataset provided in [23, 10] and used in VNN-COMP 2020 [22]. The benchmark 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 [7], a basic BaB and LP based verifier; (2) MIPplanet [15], a customized MIP solver for NN verification where unstable ReLU neurons are randomly selected for splitting; (3) ERAN [35, 33, 36, 34], an abstract interpretation based verifier which performs well on this benchmark in VNN-COMP 2020; (4) GNN-Online [23], a BaB and LP based verifiers using a learned Graph Neural Network (GNN) to guide the ReLU splits; (5) BDD+ BaBSR [6], a verification framework based on Lagrangian decomposition on GPUs (BDD+) with BaBSR branching strategy; (6) OVAL (BDD+ GNN) [6, 23], a strong verifier in VNN-COMP 2020 using BDD+ with GNN guiding the ReLU splits; (7) A.set BaBSR and (8) Big-M+A.set BaBSR [10], very recent dual-space verifiers on GPUs with a tighter linear relaxation than triangle LP relaxations; (9) Fast-and-Complete [45],

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

	CIFAR-10 Base			CIFAR-10 Wide			CIFAR-10 Deep		
Method	time(s)	branches	%timeout	time(s)	branches	%timeout	time(s)	branches	%timeout
BaBSR [7]	2367.78	1020.55	36.00	2871.14	812.65	49.00	2750.75	401.28	39.00
MIPplanet [15]	2849.69	-	68.00	2417.53	-	46.00	2302.25	-	40.00
ERAN* [35, 33, 36, 34]	805.94	-	5.00	632.20	-	9.00	545.72	-	**0.00**
GNN-online [23]	1794.85	565.13	33.00	1367.38	372.74	15.00	1055.33	131.85	4.00
BDD+ BaBSR [6]	807.91	195480.14	20.00	505.65	74203.11	10.00	266.28	12722.74	4.00
OVAL (BDD+ GNN)* [6, 23]	662.17	67938.38	16.00	280.38	17895.94	6.00	94.69	1990.34	1.00
A.set BaBSR [10]	381.78	12004.60	7.00	165.91	2233.10	3.00	190.28	2491.55	2.00
BigM+A.set BaBSR [10]	390.44	11938.75	7.00	172.65	4050.59	3.00	177.22	3275.25	2.00
Fast-and-Complete [45]	695.01	119522.65	17.00	495.88	80519.85	9.00	105.64	2455.11	1.00
BaDNB (BDD+ FSB)[11]	309.29	38239.04	7.00	165.53	11214.44	4.00	10.50	368.16	**0.00**
β-CROWN BaBSR	226.06	509608.50	6.00	118.26	217691.24	3.00	6.12	204.66	**0.00**
β-CROWN FSB	**118.23**	208018.21	**3.00**	**78.32**	116912.57	**2.00**	**5.69**	41.12	**0.00**

* OVAL (BDD+ GNN) and ERAN results are from VNN-COMP 2020 report [22]. Other results were reported by their authors.

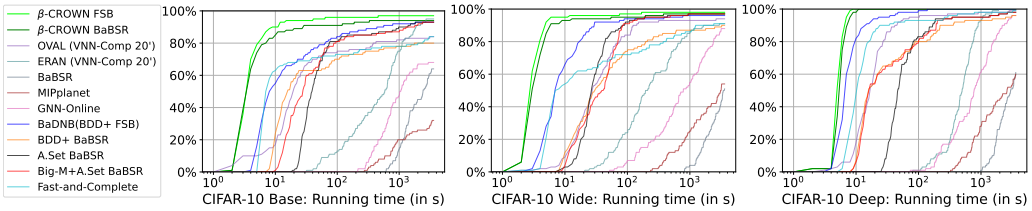


Figure 1: Percentage of solved properties with growing running time. β-CROWN FSB (light green) and β-CROWN BaBSR (dark green) clearly lead in all 3 settings and solve over 90% properties within 10 seconds.

which uses CROWN (LiRPA) on GPUs as the incomplete verifier in BaB without neuron split constraints; (10) BaDNB (BDD+ FSB) [11], 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 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 [10], Fast-and-Complete [45] and BaDNB [11]. Our benefits are more clearly shown in Figure 1, where we solve over 90% examples under 10s and most other verifiers can verify much less or none of the properties within 10s. We see a 2 to 3 orders of magitudes speedup in Figure 1 compared to CPU based verifiers such as MIPplanet and BaBSR.

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 and 4 MLP networks for MNIST and CIFAR-10 under the same set of 1000 images and perturbation ϵ as reported in [35, 37, 26]. Among the baselines, kPoly [35], OptC2V [37] and PRIMA [26] utilize state-of-the-art multi-neuron linear relaxation for ReLUs and can bypass the single-neuron convex relaxation barrier [30], and are among the strongest incomplete verifiers. β-CROWN FSB achieves better verified accuracy on all models using a similar or less amount of time. Some models, such as MNIST ConvBig and CIFAR ConvBig, 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 [9], 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 [28]. The SDP formulation can be 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 10 labels each. Due to resource limitations, we directly quote SDP-FO results from [9] on the same set of models. We evaluate verified accuracy on the same set of 200 test images for other baselines. We include a concurrent work PRIMA [26], the strongest multi-neuron linear relaxation baseline in Table 2, which generally outperforms kPoly and OptC2V. Table 3 shows that overall we are 3 orders of magnitude faster than SDP-FO while still achieving consistently higher 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 [9]. Specifically, we use the verification objective

Table 2: **Verified accuracy (%)** and avg. time (s) of 1000 images evaluated on the ERAN models in [35, 37, 26]. kPoly, OptC2V and PRIMA are strong incomplete verifiers that can break the convex relaxation barrier [30]. The average time reported by us excludes examples that are classified incorrectly.

Dataset (Same settings as [35, 37, 26])	Model	CROWN/DeepPoly*[36] Verified%	CROWN/DeepPoly*[36] Time (s)	kPoly [35] Ver.%	kPoly [35] Time(s)	OptC2V [37] Ver.%	OptC2V [37] Time(s)	PRIMA† [26] Ver.%	PRIMA† [26] Time(s)	β-CROWN FSB Ver.%	β-CROWN FSB Time(s)	Upper bound
MNIST	MLP 5×100‡	16.0	0.7	44.1	307	42.9	137	51.0	159	**69.9**	102	84.2
	MLP 8×100	18.2	1.4	36.9	171	38.4	759	42.8	301	**62.0**	103	82.0
	MLP 5×200	29.2	2.4	57.4	187	60.1	403	69.0	224	**77.4**	86	90.1
	MLP 8×200	25.9	5.6	50.6	464	52.8	3451	62.4	395	**73.5**	95	91.1
	ConvSmall	15.8	3	34.7	477	43.6	55	59.8	42	**72.7**	7.0	73.2
	ConvBig	71.1	21	73.6	40	77.1	102	77.5	11	**79.3**	3.1	80.4
CIFAR	ConvSmall	35.9	4	39.9	86	39.8	105	44.6	13	**46.3**	6.8	48.1
	ConvBig	42.1	43	45.9	346	No public code		48.3	176	**51.6**	15.3	55.0
	ResNet	24.1	1	24.5	91	cannot run		**24.8**	1.7	**24.8**	1.6	24.8

* CROWN/DeepPoly evaluated on CPU. † PRIMA is a concurrent work and its results are from [26] (Oct 26, 2021 version), except that ResNet results are from personal communications with the authors due to a different input normalization used. ‡ Because these MLP models are fairly small, some of their intermediate layer bounds are computed by mixed integer programming (MIP) using 80% time budget before branch and bound starts and β-CROWN FSB is used during the branch and bound process. We find that tighter intermediate bounds by MIP is beneficial for these small MLP models.

Table 3: **Verified accuracy (%)** and avg. per-example verification time (s) on 7 models from SDP-FO [9]. 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, a concurrent work, is the state-of-the-art relaxation barrier breaking method; 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%	CROWN/DeepPoly Time (s)	SDP-FO [9]* Ver.%	SDP-FO [9]* Time(s)	PRIMA [26] Ver.%	PRIMA [26] Time(s)	β-CROWN FSB Ver.%	β-CROWN FSB Time(s)	Upper bound
MNIST	CNN-A-Adv	1.0	0.1	43.4	>20h	44.5	135.9	**70.5**	21.1	76.5
CIFAR	CNN-B-Adv	21.5	0.5	32.8	>25h	38.0	343.6	**46.5**	32.2	65.0
	CNN-B-Adv-4	43.5	0.9	46.0	>25h	53.5	43.8	**54.0**	11.6	63.5
	CNN-A-Adv	35.5	0.6	39.6	>25h	41.5	4.8	**44.0**	5.8	50.0
	CNN-A-Adv-4	41.5	0.7	40.0	>25h	45.0	4.9	**46.0**	5.6	49.5
	CNN-A-Mix	23.5	0.4	39.6	>25h	37.5	34.3	**41.5**	49.6	53.0
	CNN-A-Mix-4	38.0	0.5	47.8	>25h	48.5	7.0	**50.5**	5.9	57.5

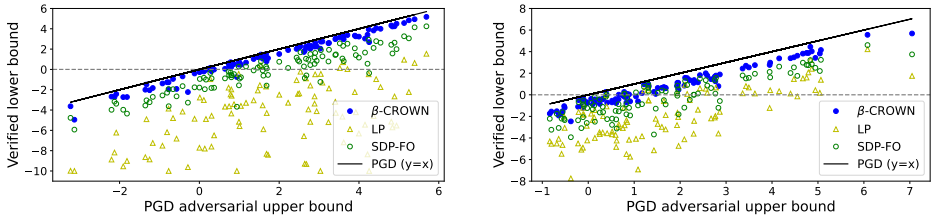
* SDP-FO results are directly from their paper due to its very long running time (>20h per example). † PRIMA experiments were done using commit 396dc7a, released on June 4, 2021. PRIMA and β-CROWN FSB results are on the same set of 200 examples (first 200 examples of CIFAR-10 dataset) and we don't run verifiers on examples that are classified incorrectly or can be attacked by a 200-step PGD. β-CROWN uses 1 GPU and 1 CPU; PRIMA uses 1 GPU and 20 CPUs.

$f(x) := z_y^{(L)}(x) - z_{y'}^{(L)}(x)$, where $z^{(L)}$ is the logit layer output, y and y' are the true label and the runner-up label. For each test image, a 200-step PGD attack [24] provides 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 3 minutes while SDP-FO needs 178 minutes on average. LP verifier with triangle relaxations produces much looser bounds than β-CROWN FSB and SDP-FO. Additional results are in Appendix C.2.

VNN-COMP 2021 results. We encourage the readers to checkout the report of the Second International Verification of Neural Networks Competition (VNN-COMP 2021) [3] with 9 additional benchmarks and 12 competing methods evaluated in a standardized testing environment on AWS. Our entry α,β-CROWN is based on the β-CROWN algorithm in this work and uses the same codebase.

5 Related Work

Many early complete verifiers for neural networks relied on existing solvers such as MILP or SMT solvers [20, 15, 19, 12, 38] and were limited to very small problem instances. Branch and bound (BaB)



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

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 2 to 3 hours per point.

based method was proposed to better exploit the network structure using LP-based incomplete verifier for bounding and ReLU splits for branching [8, 40, 23, 5]. Besides branching on ReLU neurons, input domain branching was also considered in [41, 29, 1] but limited by input dimensions [8].

Recently, a few approaches have been proposed to use efficient iterative solvers or bound propagation methods on GPUs without relying on LP solvers. Bunel et al. [6] 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. [10] used a dual-space verifier with a linear relaxation [2, 37] tighter than triangle LP relaxation, but in most settings the extra computational costs and difficulties for an efficient implementation offset its benefits (more discussions in section B.2). A concurrent work BaDNB [11] proposed a new branching strategy, filtered smart branching (FSB), combined with Lagrangian decomposition to get better verification performance. Xu et al. [44] 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. [30] shows the inherent limitation of using per-neuron convex relaxations for verification problems. Singh et al. [35] and Müller et al. [26] broke this barrier by considering constraints involving multiple ReLU neurons; Tjandraatmadja et al. [37] proposed to relax a linear layer with a ReLU neuron together using a strong mixed-integer programming formulation [1]. SDP based relaxations [28, 16, 14] typically produce tight bounds but with significantly higher cost. The most recent GPU based SDP verifier [9] is still relatively slow and can take 2 hours to verify a single image. In this work, we impose neuron split constraints using β-CROWN and combine it with branch and bound done in parallel on GPUs. Although for each subdomain in BaB, β-CROWN is still subject to the convex relaxation barrier, the efficiency of β-CROWN BaB allows it to quickly explore a very large number of subdomains and outperform existing convex barrier breaking 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 [42, 39, 25, 43, 18, 25, 47, 4, 32]. 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.

Limitations. Our verifier has several limitations which are commonly shared by most existing BaB-based complete verifiers. First, we focused on ReLU which can be split into two linear cases. For other non-piecewise linear activation functions, although it is still possible to conduct branch and bound, it is difficult to guarantee completeness. Second, we discussed only the norm perturbations for input domains. In practice, the threat model may involve complicated and nonconvex perturbation specifications. Third, although our GPU accelerated verifier outperforms existing ones, all BaB based verifiers, including ours, are still limited to relatively small models far from the ImageNet scale. Finally, we have only demonstrated robustness verification of image classification tasks, and generalizing it to give verification guarantees for other tasks such as robust deep reinforcement learning [27, 48, 49] is an interesting direction for future work.

Societal Impact NNs have been used in an increasingly wide range of real-world applications and play an important role in artificial intelligence (AI). The trustworthiness and robustness of NNs have become crucial factors since AI plays an important role in modern society. β-CROWN is a strong neural network verifier which can be used to check certain properties of neural networks, which can be helpful for guaranteeing the robustness, correctness, and fairness of NNs in applications that can directly or indirectly impact human life. We believe our work has overall positive societal impacts, although it may potentially be misused to identify the weakness of NNs and guide attacks.

Acknowledgement

This work is supported by NSF grant CNS18-01426; an ARL Young Investigator (YIP) award; an NSF CAREER award; a Google Faculty Fellowship; a Capital One Research Grant; and a J.P. Morgan Faculty Award; Air Force Research Laboratory under FA8750-18-2-0058; NSF IIS-1901527, NSF IIS-2008173 and NSF CAREER-2048280; and NSF CNS-1932351. Huan Zhang is supported by funding from the Bosch Center for Artificial Intelligence.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

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

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

[24] 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.

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

[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. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary. Robust deep reinforcement learning with adversarial attacks. *arXiv preprint arXiv:1712.03632*, 2017.

[28] 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.

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

[30] 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.

[31] 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.

[32] Z. Shi, Y. Wang, H. Zhang, J. Yi, and C.-J. Hsieh. Fast certified robust training via better initialization and shorter warmup. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

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

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

[40] 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.

[41] 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.

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

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] H. Zhang, H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C.-J. Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[49] H. Zhang, H. Chen, D. Boning, and C.-J. Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *International Conference on Learning Representations (ICLR)*, 2021.

A Proofs for β-CROWN

A.1 Proofs for deriving β-CROWN using bound propagation

Lemma 2.1 is from part of the proof of the main theorem in Zhang et al. [46]. Here we present it separately to use it as an useful subprocedure for our later proofs.

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

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

where $\mathbf{D}$ is a diagonal matrix defined as:

$$\mathbf{D}_{j,j} = \begin{cases} 1, & \text{if } \mathbf{l}_j \geq 0 \\ 0, & \text{if } \mathbf{u}_j \leq 0 \\ \boldsymbol{\alpha}_j, & \text{if } \mathbf{u}_j > 0 > \mathbf{l}_j \text{ and } w_j \geq 0 \\ \frac{\mathbf{u}_j}{\mathbf{u}_j - \mathbf{l}_j}, & \text{if } \mathbf{u}_j > 0 > \mathbf{l}_j \text{ and } w_j < 0, \end{cases} \tag{1}$$

$0 \leq \boldsymbol{\alpha}_j \leq 1$ are free variables, $b' = w^\top \underline{\mathbf{b}}$ and each element in $\underline{\mathbf{b}}$ is

$$\underline{\mathbf{b}}_j = \begin{cases} 0, & \text{if } \mathbf{l}_j > 0 \text{ or } \mathbf{u}_j \leq 0 \\ 0, & \text{if } \mathbf{u}_j > 0 > \mathbf{l}_j \text{ and } w_j \geq 0 \\ -\frac{\mathbf{u}_j \mathbf{l}_j}{\mathbf{u}_j - \mathbf{l}_j}, & \text{if } \mathbf{u}_j > 0 > \mathbf{l}_j \text{ and } w_j < 0. \end{cases} \tag{2}$$

Proof. For the j-th ReLU neuron, if $\mathbf{l}_j \geq 0$, then $\mathrm{ReLU}(v_j) = v_j$; if $\mathbf{u}_j < 0$, then $\mathrm{ReLU}(v_j) = 0$. For the case of $\mathbf{l}_j < 0 < \mathbf{u}_j$, the ReLU function can be linearly upper and lower bounded within this range:

$$\boldsymbol{\alpha}_j v_j \leq \mathrm{ReLU}(v_j) \leq \frac{\mathbf{u}_j}{\mathbf{u}_j - \mathbf{l}_j}(v_j - \mathbf{l}_j) \quad \forall\, \mathbf{l}_j \leq v_j \leq \mathbf{u}_j$$

where $0 \leq \boldsymbol{\alpha}_j \leq 1$ is a free variable - any value between 0 and 1 produces a valid lower bound. To lower bound $w^\top \mathrm{ReLU}(v) = \sum_j w_j \mathrm{ReLU}(v_j)$, for each term in this summation, we take the lower bound of $\mathrm{ReLU}(v_j)$ if w_j is positive and take the upper bound of $\mathrm{ReLU}(v_j)$ if w_j is negative (reflected in the definitions of $\mathbf{D}$ and $\underline{\mathbf{b}}$). This conservative choice allows us to always obtain a lower bound $\forall\, \mathbf{l} \leq v \leq \mathbf{u}$:

$$\sum_j w_j \mathrm{ReLU}(v_j) \geq \sum_j w_j \left(\mathbf{D}_{j,j} v_j + \underline{\mathbf{b}}_j\right) = w^\top \mathbf{D} v + w^\top \underline{\mathbf{b}} = w^\top \mathbf{D} v + b'$$

where $\mathbf{D}_{j,j}$ and $\underline{\mathbf{b}}_j$ are defined in Eq. 1 and Eq. 2 representing the lower or upper bounds of ReLU. □

Before proving our main theorem (Theorem 3.1), we first define matrix $\boldsymbol{\Omega}$, which is the product of a series of model weights $\mathbf{W}$ and "weights" for relaxed ReLU layers $\mathbf{D}$:

Definition A.1. *Given a set of matrices $\mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(L)}$ and $\mathbf{D}^{(1)}, \cdots, \mathbf{D}^{(L-1)}$, we define a recursive function $\boldsymbol{\Omega}(k, i)$ for $1 \leq i \leq k \leq L$ as*

$$\boldsymbol{\Omega}(i, i) = \boldsymbol{I},\ \boldsymbol{\Omega}(k{+}1, i) = \mathbf{W}^{(k+1)} \mathbf{D}^{(k)} \boldsymbol{\Omega}(k, i)$$

For example, $\boldsymbol{\Omega}(3,1) = \mathbf{W}^{(3)}\mathbf{D}^{(2)}\mathbf{W}^{(2)}\mathbf{D}^{(1)}$, $\boldsymbol{\Omega}(5,2) = \mathbf{W}^{(5)}\mathbf{D}^{(4)}\mathbf{W}^{(4)}\mathbf{D}^{(3)}\mathbf{W}^{(3)}\mathbf{D}^{(2)}$. Now we present our main theorem with each term explicitly written:

Theorem 3.1 (β-CROWN bound)**.** *Given a L-layer neural network $f(x) : \mathbb{R}^{d_0} \to \mathbb{R}$ 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 constraint $\mathcal{C}$ and split constraint $\mathcal{Z}$. We have*

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) \geq \max_{\boldsymbol{\beta} \geq 0} \min_{x \in \mathcal{C}} (\boldsymbol{a} + \mathbf{P}\boldsymbol{\beta})^\top x + \mathbf{q}^\top \boldsymbol{\beta} + c, \tag{3}$$

where $\mathbf{P} \in \mathbb{R}^{d_0 \times (\sum_{i=1}^{L-1} d_i)}$ is a matrix containing blocks $\mathbf{P} := \left[\mathbf{P}_1{}^\top\ \mathbf{P}_2{}^\top\ \cdots\ \mathbf{P}_{L-1}{}^\top\right]$, $\mathbf{q} \in \mathbb{R}^{\sum_{i=1}^{L-1} d_i}$ is a vector $\mathbf{q} := \left[\mathbf{q}_1{}^\top\ \cdots\ \mathbf{q}_{L-1}{}^\top\right]^\top$, and each term is defined as:

$$\boldsymbol{a} = \left[\boldsymbol{\Omega}(L, 1)\mathbf{W}^{(1)}\right]^\top \in \mathbb{R}^{d_0 \times 1} \tag{4}$$

$$\mathbf{P}_i = \mathbf{S}^{(i)}\mathbf{\Omega}(i,1)\mathbf{W}^{(1)} \in \mathbb{R}^{d_i \times d_0}, \quad \forall\, 1 \le i \le L-1 \tag{5}$$

$$\mathbf{q}_i = \sum_{k=1}^{i} \mathbf{S}^{(i)}\mathbf{\Omega}(i,k)\mathbf{b}^{(k)} + \sum_{k=2}^{i} \mathbf{S}^{(i)}\mathbf{\Omega}(i,k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} \in \mathbb{R}^{d_i}, \quad \forall\, 1 \le i \le L-1 \tag{6}$$

$$c = \sum_{i=1}^{L} \mathbf{\Omega}(L,i)\mathbf{b}^{(i)} + \sum_{i=2}^{L} \mathbf{\Omega}(L,i)\mathbf{W}^{(i)}\underline{\mathbf{b}}^{(i-1)} \tag{7}$$

diagonal matrices $\mathbf{D}^{(i)}$ *and vector* $\underline{\mathbf{b}}^{(i)}$ *are determined by the relaxation of ReLU neurons, and* $\mathbf{A}^{(i)} \in \mathbb{R}^{1\times d_i}$ *represents the linear relationship between* $f(x)$ *and* $\hat{z}^{(i)}$*.* $\mathbf{D}^{(i)}$ *and* $\underline{\mathbf{b}}^{(i)}$ *depend on* $\mathbf{A}^{(i)}$*,* $\mathbf{l}^{(i)}$ *and* $\mathbf{u}^{(i)}$*:*

$$\mathbf{D}^{(i)}_{j,j} = \begin{cases} 1, & \text{if } \mathbf{l}^{(i)}_j \ge 0 \text{ or } j \in \mathcal{Z}^{+(i)} \\ 0, & \text{if } \mathbf{u}^{(i)}_j \le 0 \text{ or } j \in \mathcal{Z}^{-(i)} \\ \boldsymbol{\alpha}_j, & \text{if } \mathbf{u}^{(i)}_j > 0 > \mathbf{l}^{(i)}_j \text{ and } j \notin \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)} \text{ and } \mathbf{A}^{(i)}_{1,j} \ge 0 \\ \frac{\mathbf{u}_j}{\mathbf{u}_j - \mathbf{l}_j}, & \text{if } \mathbf{u}^{(i)}_j > 0 > \mathbf{l}^{(i)}_j \text{ and } j \notin \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)} \text{ and } \mathbf{A}^{(i)}_{1,j} < 0 \end{cases} \tag{8}$$

$$\underline{\mathbf{b}}^{(i)}_j = \begin{cases} 0, & \text{if } \mathbf{l}^{(i)}_j > 0 \text{ or } \mathbf{u}^{(i)}_j \le 0 \text{ or } j \in \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)} \\ 0, & \text{if } \mathbf{u}^{(i)}_j > 0 > \mathbf{l}^{(i)}_j \text{ and } j \notin \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)} \text{ and } \mathbf{A}^{(i)}_{1,j} \ge 0 \\ -\frac{\mathbf{u}^{(i)}_j \mathbf{l}^{(i)}_j}{\mathbf{u}^{(i)}_j - \mathbf{l}^{(i)}_j}, & \text{if } \mathbf{u}^{(i)}_j > 0 > \mathbf{l}^{(i)}_j \text{ and } j \notin \mathcal{Z}^{+(i)} \cup \mathcal{Z}^{-(i)} \text{ and } \mathbf{A}^{(i)}_{1,j} < 0 \end{cases} \tag{9}$$

$$\mathbf{A}^{(i)} = \begin{cases} \mathbf{W}^{(L)}, & i = L-1 \\ (\mathbf{A}^{(i+1)}\mathbf{D}^{(i+1)} + \boldsymbol{\beta}^{(i+1)\top}\mathbf{S}^{(i+1)})\mathbf{W}^{(i+1)}, & 0 \le i \le L-2 \end{cases} \tag{10}$$

Proof. We prove this theorem by induction: assuming we know the bounds with respect to layer $\hat{z}^{(m)}$, we derive bounds for $\hat{z}^{(m-1)}$ until we reach $m = 0$ and by definition $\hat{z}^{(0)} = x$. We first define a set of matrices and vectors $\boldsymbol{a}^{(m)}$, $\mathbf{P}^{(m)}$, $\mathbf{q}^{(m)}$, $c^{(m)}$, where $\mathbf{P}^{(m)} \in \mathbb{R}^{d_m \times (\sum_{i=m+1}^{L-1} d_i)}$ is a matrix containing blocks $\mathbf{P} := \begin{bmatrix} \mathbf{P}^{(m)}_{m+1}{}^{\top} & \cdots & \mathbf{P}^{(m)}_{L-1}{}^{\top} \end{bmatrix}$, $\mathbf{q} \in \mathbb{R}^{\sum_{i=m+1}^{L-1} d_i}$ is a vector $\mathbf{q} := \begin{bmatrix} \mathbf{q}^{(m)}_{m+1}{}^{\top} & \cdots & \mathbf{q}^{(m)}_{L-1}{}^{\top} \end{bmatrix}^{\top}$, and each term is defined as:

$$\boldsymbol{a}^{(m)} = \left[\mathbf{\Omega}(L, m+1)\mathbf{W}^{(m+1)}\right]^{\top} \in \mathbb{R}^{d_m \times 1} \tag{11}$$

$$\mathbf{P}^{(m)}_i = \mathbf{S}^{(i)}\mathbf{\Omega}(i, m+1)\mathbf{W}^{(m+1)} \in \mathbb{R}^{d_i \times d_m}, \quad \forall\, m+1 \le i \le L-1 \tag{12}$$

$$\mathbf{q}^{(m)}_i = \sum_{k=m+1}^{i} \mathbf{S}^{(i)}\mathbf{\Omega}(i,k)\mathbf{b}^{(k)} + \sum_{k=m+2}^{i} \mathbf{S}^{(i)}\mathbf{\Omega}(i,k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} \in \mathbb{R}^{d_m}, \quad \forall\, m+1 \le i \le L-1 \tag{13}$$

$$c^{(m)} = \sum_{i=m+1}^{L} \mathbf{\Omega}(L,i)\mathbf{b}^{(i)} + \sum_{i=m+2}^{L} \mathbf{\Omega}(L,i)\mathbf{W}^{(i)}\underline{\mathbf{b}}^{(i-1)} \tag{14}$$

and we claim that

$$\min_{\substack{x\in\mathcal{C} \\ z\in\mathcal{Z}}} f(x) \ge \max_{\tilde{\boldsymbol{\beta}}^{(m+1)} \ge 0} \min_{\substack{x\in\mathcal{C} \\ \hat{z}^{(m)} \in \hat{\mathcal{Z}}^{(m)}}} (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^{\top}\hat{z}^{(m)} + \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)} \tag{15}$$

where $\tilde{\boldsymbol{\beta}}^{(m+1)} := \left[\boldsymbol{\beta}^{(m+1)\top} \cdots \boldsymbol{\beta}^{(L-1)\top}\right]^{\top}$ concatenating all $\boldsymbol{\beta}^{(i)}$ variables up to layer $m+1$.

For the base case $m = L - 1$, we simply have

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

No maximization is needed and $\boldsymbol{a}^{(m)} = \left[\boldsymbol{\Omega}(L,L)\mathbf{W}^{(L)}\right]^\top = \mathbf{W}^{(L)\top}$, $c^{(m)} = \sum_{i=L}^{L} \boldsymbol{\Omega}(L,i)\mathbf{b}^{(i)} = \mathbf{b}^{(L)}$. Other terms are zero.

In Section 3.1 we have shown the intuition of the proof by demonstrating how to derive the bounds from layer $\hat{z}^{(L-1)}$ to $\hat{z}^{(L-2)}$. The case for $m = L - 2$ is presented in Eq. 6.

Now we show the induction from $\hat{z}^{(m)}$ to $\hat{z}^{(m-1)}$. Starting from Eq. 15, since $\hat{z}^{(m)} = \mathrm{ReLU}(z^{(m)})$ we apply Lemma 2.1 by setting $w = \left[\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)}\right]^\top := \mathbf{A}^{(m)}$. It is easy to show that $\mathbf{A}^{(m)}$ can also be equivalently and recursively defined in Eq. 10 (see Lemma A.2). Based on Lemma 2.1 we have $\mathbf{D}^{(m)}$ and $\underline{\mathbf{b}}^{(m)}$ defined as in Eq. 8 and Eq. 9, so Eq. 15 becomes

$$\begin{aligned}\min_{\substack{x\in\mathcal{C}\\ z\in\mathcal{Z}}} f(x) \geq \max_{\tilde{\boldsymbol{\beta}}^{(m+1)}\geq 0} \min_{\substack{x\in\mathcal{C}\\ \tilde{z}^{(m)}\in\tilde{\mathcal{Z}}^{(m)}}} & (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \mathbf{D}^{(m)} z^{(m)} \\ & + (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \underline{\mathbf{b}}^{(m)} + \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)}\end{aligned} \tag{16}$$

Note that when we apply Lemma 2.1, for $j \in \mathcal{Z}^{+(i)}$ (positive split) we simply treat the neuron j as if $\mathbf{l}_j^{(i)} \geq 0$, and for $j \in \mathcal{Z}^{-(i)}$ (negative split) we simply treat the neuron j as if $\mathbf{u}_j^{(i)} \leq 0$. Now we add the multiplier $\boldsymbol{\beta}^{(m)}$ to $z^{(m)}$ to enforce per-neuron split constraints:

$$\begin{aligned}\min_{\substack{x\in\mathcal{C}\\ z\in\mathcal{Z}}} f(x) &\geq \max_{\tilde{\boldsymbol{\beta}}^{(m+1)}\geq 0} \min_{\substack{x\in\mathcal{C}\\ \tilde{z}^{(m-1)}\in\tilde{\mathcal{Z}}^{(m-1)}}} \max_{\boldsymbol{\beta}^{(m)}\geq 0} \quad (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \mathbf{D}^{(m)} z^{(m)} + \boldsymbol{\beta}^{(m)\top}\mathbf{S}^{(m)} z^{(m)} \\ &\qquad\qquad + (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \underline{\mathbf{b}}^{(m)} + \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)} \\ &\geq \max_{\tilde{\boldsymbol{\beta}}^{(m)}\geq 0} \min_{\substack{x\in\mathcal{C}\\ \tilde{z}^{(m-1)}\in\tilde{\mathcal{Z}}^{(m-1)}}} \quad (\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)} + \tilde{\boldsymbol{\beta}}^{(m+1)\top}\mathbf{P}^{(m)\top}\mathbf{D}^{(m)} + \boldsymbol{\beta}^{(m)\top}\mathbf{S}^{(m)}) z^{(m)} \\ &\qquad\qquad + (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \underline{\mathbf{b}}^{(m)} + \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)}\end{aligned}$$

Similar to what we did in Eq. 5, we swap the $\min$ and $\max$ in the second inequality due to weak duality, such that every maximization on $\boldsymbol{\beta}^{(i)}$ is before $\min$. Then, we substitute $\hat{z}^{(m)} = \mathbf{W}^{(m)}\hat{z}^{(m-1)} + \mathbf{b}^{(m)}$ and obtain:

$$\begin{aligned}\min_{\substack{x\in\mathcal{C}\\ z\in\mathcal{Z}}} f(x) \geq \max_{\tilde{\boldsymbol{\beta}}^{(m)}\geq 0} \min_{\substack{x\in\mathcal{C}\\ \tilde{z}^{(m-1)}\in\tilde{\mathcal{Z}}^{(m-1)}}} & (\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)} + \tilde{\boldsymbol{\beta}}^{(m+1)\top}\mathbf{P}^{(m)\top}\mathbf{D}^{(m)} + \boldsymbol{\beta}^{(m)\top}\mathbf{S}^{(m)})^\top \mathbf{W}^{(m)}\hat{z}^{(m-1)} \\ & + (\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)} + \tilde{\boldsymbol{\beta}}^{(m+1)\top}\mathbf{P}^{(m)\top}\mathbf{D}^{(m)} + \boldsymbol{\beta}^{(m)\top}\mathbf{S}^{(m)})^\top \mathbf{b}^{(m)} \\ & + (\boldsymbol{a}^{(m)} + \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)})^\top \underline{\mathbf{b}}^{(m)} + \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)} \\ = & \left[\underbrace{\left[\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)}\mathbf{W}^{(m)}\right]^\top}_{\boldsymbol{a}'} + \underbrace{(\tilde{\boldsymbol{\beta}}^{(m+1)\top}\mathbf{P}^{(m)\top}\mathbf{D}^{(m)}\mathbf{W}^{(m)} + \boldsymbol{\beta}^{(m)\top}\mathbf{S}^{(m)}\mathbf{W}^{(m)})}_{\mathbf{P}'\tilde{\boldsymbol{\beta}}^{(m)}}\right]^\top \hat{z}^{(m-1)} \\ & + \underbrace{\left((\mathbf{P}^{(m)\top}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \mathbf{P}^{(m)\top}\underline{\mathbf{b}}^{(m)} + \mathbf{q}^{(m)})^\top\tilde{\boldsymbol{\beta}}^{(m+1)} + (\mathbf{S}^{(m)}\mathbf{b}^{(m)})^\top\boldsymbol{\beta}^{(m)}\right)}_{\mathbf{q}'^\top\tilde{\boldsymbol{\beta}}^{(m)}} \\ & + \underbrace{\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \boldsymbol{a}^{(m)\top}\underline{\mathbf{b}}^{(m)} + c^{(m)}}_{c'}\end{aligned}$$

Now we evaluate each term $\boldsymbol{a}'$, $\mathbf{P}'$, $\mathbf{q}'$ and c' and show the induction holds. For $\boldsymbol{a}'$ and $\mathbf{q}'$ we have:

$$\boldsymbol{a}' = \left[\boldsymbol{a}^{(m)\top}\mathbf{D}^{(m)}\mathbf{W}^{(m)}\right]^\top = \left[\boldsymbol{\Omega}(L, m+1)\mathbf{W}^{(m+1)}\mathbf{D}^{(m)}\mathbf{W}^{(m)}\right]^\top = \left[\boldsymbol{\Omega}(L, m)\mathbf{W}^{(m)}\right]^\top = \boldsymbol{a}^{(m-1)}$$

$$\begin{aligned}
c' &= c^{(m)} + \boldsymbol{\Omega}(L, m+1)\mathbf{W}^{(m+1)}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \boldsymbol{\Omega}(L, m+1)\mathbf{W}^{(m+1)}\underline{\mathbf{b}}^{(m)} \\
&= \sum_{i=m+1}^{L} \boldsymbol{\Omega}(L, i)\mathbf{b}^{(i)} + \sum_{i=m+2}^{L} \boldsymbol{\Omega}(L, i)\mathbf{W}^{(i)}\underline{\mathbf{b}}^{(i-1)} + \boldsymbol{\Omega}(L, m)\mathbf{b}^{(m)} + \boldsymbol{\Omega}(L, m+1)\mathbf{W}^{(m+1)}\underline{\mathbf{b}}^{(m)} \\
&= \sum_{i=m}^{L} \boldsymbol{\Omega}(L, i)\mathbf{b}^{(i)} + \sum_{i=m+1}^{L} \boldsymbol{\Omega}(L, i)\mathbf{W}^{(i)}\underline{\mathbf{b}}^{(i-1)} \\
&= c^{(m-1)}
\end{aligned}$$

For $\mathbf{P}' := \left[\mathbf{P}'_m{}^\top \cdots \mathbf{P}'_{L-1}{}^\top\right]$, we have a new block $\mathbf{P}'_m$ where

$$\mathbf{P}'_m = \mathbf{S}^{(m)}\mathbf{W}^{(m)} = \mathbf{S}^{(m)}\boldsymbol{\Omega}(m, m)\mathbf{W}^{(m)} = \mathbf{P}_m^{(m-1)}$$

for other blocks where $m+1 \leq i \leq L-1$,

$$\mathbf{P}'_i = \mathbf{P}_i^{(m)}\mathbf{D}^{(m)}\mathbf{W}^{(m)} = \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m+1)\mathbf{W}^{(m+1)}\mathbf{D}^{(m)}\mathbf{W}^{(m)} = \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m)\mathbf{W}^{(m)} = \mathbf{P}_i^{(m-1)}$$

For $\mathbf{q}' := \left[\mathbf{q}'_m{}^\top \cdots \mathbf{q}'_{L-1}{}^\top\right]$, we have a new block $\mathbf{q}'_m$ where

$$\mathbf{q}'_m = \mathbf{S}^{(m)}\mathbf{b}^{(m)} = \sum_{k=m}^{m} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{b}^{(i)} = \mathbf{q}_m^{(m-1)}$$

for other blocks where $m+1 \leq i \leq L-1$,

$$\begin{aligned}
\mathbf{q}'_i &= \mathbf{q}_i^{(m)} + \mathbf{P}^{(m)\top}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \mathbf{P}^{(m)\top}\underline{\mathbf{b}}^{(m)} \\
&= \sum_{k=m+1}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{b}^{(k)} + \sum_{k=m+2}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} + \mathbf{P}^{(m)\top}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \mathbf{P}^{(m)\top}\underline{\mathbf{b}}^{(m)} \\
&= \sum_{k=m+1}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{b}^{(k)} + \sum_{k=m+2}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} \\
&\quad + \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m+1)\mathbf{W}^{(m+1)}\mathbf{D}^{(m)}\mathbf{b}^{(m)} + \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m+1)\mathbf{W}^{(m+1)}\underline{\mathbf{b}}^{(k)} \\
&= \sum_{k=m+1}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{b}^{(k)} + \sum_{k=m+2}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} + \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m)\mathbf{b}^{(m)} \\
&\quad + \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, m+1)\mathbf{W}^{(m+1)}\underline{\mathbf{b}}^{(m)} \\
&= \sum_{k=m}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{b}^{(k)} + \sum_{k=m+1}^{i} \mathbf{S}^{(i)}\boldsymbol{\Omega}(i, k)\mathbf{W}^{(k)}\underline{\mathbf{b}}^{(k-1)} \\
&= \mathbf{q}_i^{(m-1)}
\end{aligned}$$

Thus, $\boldsymbol{a}' = \boldsymbol{a}^{(m-1)}$, $\mathbf{P}' = \mathbf{P}^{(m-1)}$, $\mathbf{q}' = \mathbf{q}^{(m-1)}$ and $c' = c^{(m-1)}$ so the induction holds for layer $\hat{z}^{(m-1)}$:

$$\min_{\substack{x\in\mathcal{C}\\ z\in\mathcal{Z}}} f(x) \geq \max_{\tilde{\boldsymbol{\beta}}^{(m)}\geq 0} \min_{\substack{x\in\mathcal{C}\\ \hat{z}^{(m-1)}\in\hat{\mathcal{Z}}^{(m-1)}}} (\boldsymbol{a}^{(m-1)} + \mathbf{P}^{(m-1)}\tilde{\boldsymbol{\beta}}^{(m)})^\top\hat{z}^{(m-1)} + \mathbf{q}^{(m-1)\top}\tilde{\boldsymbol{\beta}}^{(m)} + c^{(m-1)} \tag{17}$$

Finally, Theorem 3.1 becomes the special case where $m = 0$ in Eq. 11, Eq. 12, Eq. 13 and Eq. 14. □

The next Lemma unveils the connection with CROWN [46] and is also useful for drawing connections to the dual problem.

Lemma A.2. *With* $\mathbf{D}$, $\underline{\mathbf{b}}$ *and* $\mathbf{A}$ *defined in Eq. 8, Eq. 9 and Eq. 10, we can rewrite Eq. 3 in Theorem 3.1 as:*

$$\min_{\substack{x\in\mathcal{C}\\ z\in\mathcal{Z}}} f(x) \geq \max_{\boldsymbol{\beta}\geq 0} \min_{x\in\mathcal{C}} \mathbf{A}^{(0)}x + \sum_{i=1}^{L-1} \mathbf{A}^{(i)}(\mathbf{D}^{(i)}\mathbf{b}^{(i)} + \underline{\mathbf{b}}^{(i)}) \tag{18}$$

where $\mathbf{A}^{(i)}$, $0 \leq i \leq L-1$ *contains variables* $\boldsymbol{\beta}$.

Proof. To prove this lemma, we simply follow the definition of $\mathbf{A}^{(i)}$ and check the resulting terms are the same as Eq. 3. For example,

$$\begin{aligned}
\mathbf{A}^{(0)} &= (\mathbf{A}^{(1)}\mathbf{D}^{(1)} + \boldsymbol{\beta}^{(1)\top}\mathbf{S}^{(1)})\mathbf{W}^{(1)} \\
&= \mathbf{A}^{(1)}\mathbf{D}^{(1)}\mathbf{W}^{(1)} + \boldsymbol{\beta}^{(1)\top}\mathbf{S}^{(1)}\mathbf{W}^{(1)} \\
&= (\mathbf{A}^{(2)}\mathbf{D}^{(2)} + \boldsymbol{\beta}^{(2)\top}\mathbf{S}^{(2)})\mathbf{W}^{(2)}\mathbf{D}^{(1)}\mathbf{W}^{(1)} + \boldsymbol{\beta}^{(1)\top}\mathbf{S}^{(1)}\mathbf{W}^{(1)} \\
&= \mathbf{A}^{(2)}\mathbf{D}^{(2)}\mathbf{W}^{(2)}\mathbf{D}^{(1)}\mathbf{W}^{(1)} + \boldsymbol{\beta}^{(2)\top}\mathbf{S}^{(2)}\mathbf{W}^{(2)}\mathbf{D}^{(1)}\mathbf{W}^{(1)} + \boldsymbol{\beta}^{(1)\top}\mathbf{S}^{(1)}\mathbf{W}^{(1)} \\
&= \cdots \\
&= \boldsymbol{\Omega}(L,1)\mathbf{W}^{(1)} + \sum_{i=1}^{L-1} \boldsymbol{\beta}^{(i)\top}\mathbf{S}^{(i)}\boldsymbol{\Omega}(i,1)\mathbf{W}^{(1)} \\
&= [\boldsymbol{a} + \mathbf{P}\boldsymbol{\beta}]^\top
\end{aligned}$$

Other terms can be shown similarly.

□

With this definition of $\mathbf{A}$, we can see Eq. 3 as a modified form of CROWN, with an extra term $\boldsymbol{\beta}^{(i+1)\top}\mathbf{S}^{(i+1)}$ added when computing $\mathbf{A}^{(i)}$. When we set $\boldsymbol{\beta} = 0$, we obtain the same bound propagation rule for $\mathbf{A}$ as in CROWN. Thus, only a small change is needed to implement β-CROWN given an existing CROWN implementation: we add $\boldsymbol{\beta}^{(i+1)\top}\mathbf{S}^{(i+1)}$ after the linear bound propagates backwards through a ReLU layer. We also have the same observation in the dual space, as we will show this connection in the next subsection.

A.2 Proofs for the connection to the dual space

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.*

Proof. To prove the Theorem 3.2, we demonstrate the detailed dual objective d_{LP} for Eq. 10, following a construction similar to the one in Wong and Kolter [42]. We first associate a dual variable for each constraint involved in Eq. 10 including dual variables $\boldsymbol{\beta}$ for the per-neuron split constraints introduced by BaB. Although it is possible to directly write the dual LP for Eq. 10, for easier understanding, we

first rewrite the original primal verification problem into its Lagrangian dual form as Eq. 19, with dual variables $\boldsymbol{\nu}, \boldsymbol{\xi}^+, \boldsymbol{\xi}^- \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\beta}$:

$$
\begin{aligned}
L(z, \hat{z}; \boldsymbol{\nu}, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\beta}) &= z^{(L)} + \sum_{i=1}^{L} {\boldsymbol{\nu}^{(i)}}^\top (z^{(i)} - \mathbf{W}^{(i)} \hat{z}^{(i-1)} - \mathbf{b}^{(i)}) \\
&+ {\boldsymbol{\xi}^+}^\top (\hat{z}^{(0)} - x_0 - \epsilon) + {\boldsymbol{\xi}^-}^\top (-\hat{z}^{(0)} + x_0 - \epsilon) \\
&+ \sum_{i=1}^{L-1} \sum_{\substack{j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)} \\ \mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)}}} \left[{\boldsymbol{\mu}_j^{(i)}}^\top (-\hat{z}_j^{(i)}) + {\boldsymbol{\gamma}_j^{(i)}}^\top (z_j^{(i)} - \hat{z}_j^{(i)}) + {\boldsymbol{\lambda}_j^{(i)}}^\top (-\mathbf{u}_j^{(i)} z_j^{(i)} + (\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}) \hat{z}_j^{(i)} + \mathbf{u}_j^{(i)} \mathbf{l}_j^{(i)}) \right] \\
&+ \sum_{i=1}^{L-1} \left[\sum_{z_j^{(i)} \in \mathcal{Z}^{-(i)}} \boldsymbol{\beta}_j^{(i)} z_j^{(i)} + \sum_{z_j^{(i)} \in \mathcal{Z}^{+(i)}} -\boldsymbol{\beta}_j^{(i)} z_j^{(i)} \right]
\end{aligned}
$$

Subject to:

$$
\boldsymbol{\xi}^+ \geq 0, \boldsymbol{\xi}^- \geq 0, \boldsymbol{\mu} \geq 0, \boldsymbol{\gamma} \geq 0, \boldsymbol{\lambda} \geq 0, \boldsymbol{\beta} \geq 0 \tag{19}
$$

The original minimization problem then becomes:

$$
\max_{\boldsymbol{\nu}, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\beta}} \min_{z, \hat{z}} L(z, \hat{z}, \boldsymbol{\nu}, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\beta})
$$

Given fixed intermediate bounds $\mathbf{l}, \mathbf{u}$, the inner minimization is a linear optimization problem and we can simply transfer it to the dual form. To further simplify the formula, we introduce notations similar to those in [42], where $\hat{\boldsymbol{\nu}}^{(i-1)} = {\mathbf{W}^{(i)}}^\top \boldsymbol{\nu}^{(i)}$ and $\boldsymbol{\alpha}_j^{(i)} = \frac{\boldsymbol{\gamma}_j^{(i)}}{\boldsymbol{\mu}_j^{(i)} + \boldsymbol{\gamma}_j^{(i)}}$. Then the dual form can be written as Eq. 20.

$$
\max_{0 \leq \boldsymbol{\alpha} \leq 1, \boldsymbol{\beta} \geq 0} g(\boldsymbol{\alpha}, \boldsymbol{\beta}), \text{ where}
$$

$$
g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\sum_{i=1}^{L} \boldsymbol{\nu}^{(i)\top} \mathbf{b}^{(i)} - \hat{\boldsymbol{\nu}}^{(0)\top} x_0 - ||\hat{\boldsymbol{\nu}}^{(0)}||_1 \cdot \epsilon + \sum_{i=1}^{L-1} \sum_{\substack{j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)} \\ \mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)}}} \mathbf{l}_j^{(i)} [\boldsymbol{\nu}_j^{(i)}]^+
$$

Subject to:

$$
\begin{aligned}
&\boldsymbol{\nu}^{(L)} = -1, \hat{\boldsymbol{\nu}}^{(i-1)} = {\mathbf{W}^{(i)}}^\top \boldsymbol{\nu}^{(i)}, \quad i \in \{1, \ldots, L\} \\
&\boldsymbol{\nu}_j^{(i)} = 0, \quad \text{when } \mathbf{u}_j^{(i)} \leq 0, i \in \{1, \ldots, L-1\} \\
&\boldsymbol{\nu}_j^{(i)} = \hat{\boldsymbol{\nu}}_j^{(i)}, \quad \text{when } \mathbf{l}_j^{(i)} \geq 0, i \in \{1, \ldots, L-1\} \\
&\left. \begin{aligned} &[\boldsymbol{\nu}_j^{(i)}]^+ = \mathbf{u}_j^{(i)} \boldsymbol{\lambda}_j^{(i)}, [\boldsymbol{\nu}_j^{(i)}]^- = \boldsymbol{\alpha}_j^{(i)} [\hat{\boldsymbol{\nu}}_j^{(i)}]^- \\ &\boldsymbol{\lambda}_j^{(i)} = \frac{[\hat{\boldsymbol{\nu}}_j^{(i)}]^+}{\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}}, \boldsymbol{\alpha}_j^{(i)} = \frac{\boldsymbol{\gamma}_j^{(i)}}{\boldsymbol{\mu}_j^{(i)} + \boldsymbol{\gamma}_j^{(i)}} \end{aligned} \right\} \text{when } \mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)}, j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)}, i \in \{1, \ldots, L-1\} \\
&\boldsymbol{\nu}_j^{(i)} = -\boldsymbol{\beta}_j^{(i)}, \quad j \in \mathcal{Z}^{-(i)}, i \in \{1, \ldots, L-1\} \\
&\boldsymbol{\nu}_j^{(i)} = \boldsymbol{\beta}_j^{(i)} + \hat{\boldsymbol{\nu}}_j^{(i)}, \quad j \in \mathcal{Z}^{+(i)}, i \in \{1, \ldots, L-1\} \\
&\boldsymbol{\mu} \geq 0, \boldsymbol{\gamma} \geq 0, \boldsymbol{\lambda} \geq 0, \boldsymbol{\beta} \geq 0, 0 \leq \boldsymbol{\alpha} \leq 1
\end{aligned} \tag{20}
$$

Similar to the dual form in [42] (our differences are highlighted in blue), the dual problem can be viewed in the form of another deep network by backward propagating $\boldsymbol{\nu}^{(L)}$ to $\hat{\boldsymbol{\nu}}^{(0)}$ following the rules in Eq. 20. If we look closely at the conditions and coefficients when backward propagating $\boldsymbol{\nu}_j^{(i)}$ for j-th ReLU at layer i in Eq. 20, we can observe that they match exactly to the propagation of

diagonal matrices $\mathbf{D}^{(i)}$, $\mathbf{S}^{(i)}$, and vector $\underline{\mathbf{b}}^{(i)}$ defined in Eq. 8 and Eq. 9. Therefore, using notations in Eq. 8 and Eq. 9 we can essentially simplify the dual LP problem in Eq. 20 to:

$$\begin{aligned}
&\boldsymbol{\nu}^{(L)} = -1, \hat{\boldsymbol{\nu}}^{(i-1)} = \mathbf{W}^{(i)\top}\boldsymbol{\nu}^{(i)}, \boldsymbol{\nu}^{(i)} = \mathbf{D}^{(i)}\hat{\boldsymbol{\nu}}^{(i)} - \boldsymbol{\beta}^{(i)}\mathbf{S}^{(i)}, i \in \{L, \cdots, 1\} \\
&\sum_{\substack{\mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)} \\ j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)}}} \mathbf{l}_j^{(i)}[\boldsymbol{\nu}_j^{(i)}]^+ = -\hat{\boldsymbol{\nu}}^{(i)T}\underline{\mathbf{b}}^{(i)}, j \in \{1, \cdots, d_i\}, i \in \{L-1, \cdots, 1\}
\end{aligned} \tag{21}$$

Then we prove the key claim for this proof with induction where $\boldsymbol{a}^{(m)}$ and $\mathbf{P}^{(m)}$ are defined in Eq. 11 and Eq. 12:

$$\hat{\boldsymbol{\nu}}^{(m)} = -\boldsymbol{a}^{(m)} - \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)} \tag{22}$$

When $m = L-1$, we can have $\hat{\boldsymbol{\nu}}^{(L-1)} = -\boldsymbol{a}^{(L-1)} - \mathbf{P}^{(L-1)}\tilde{\boldsymbol{\beta}}^{(L)} = -\left[\boldsymbol{\Omega}(L, L)\mathbf{W}^{(L)}\right]^\top - \mathbf{0} = -\mathbf{W}^{(L)\top}$ which is true according to Eq. 21.

Now we assume that $\hat{\boldsymbol{\nu}}^{(m)} = -\boldsymbol{a}^{(m)} - \mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)}$ holds, and we show that $\hat{\boldsymbol{\nu}}^{(m-1)} = -\boldsymbol{a}^{(m-1)} - \mathbf{P}^{(m-1)}\tilde{\boldsymbol{\beta}}^{(m)}$ will hold as well:

$$\begin{aligned}
\hat{\boldsymbol{\nu}}^{(m-1)} &= \mathbf{W}^{(m)\top}\left(\mathbf{D}^{(m)}\hat{\boldsymbol{\nu}}^{(m)} - \boldsymbol{\beta}^{(m)}\mathbf{S}^{(m)}\right) \\
&= -\mathbf{W}^{(m)\top}\mathbf{D}^{(m)}\boldsymbol{a}^{(m)} - \mathbf{W}^{(m)\top}\mathbf{D}^{(m)}\mathbf{P}^{(m)}\tilde{\boldsymbol{\beta}}^{(m+1)} - \mathbf{W}^{(m)\top}\boldsymbol{\beta}^{(m)}\mathbf{S}^{(m)} \\
&= -\boldsymbol{a}^{(m-1)} - \left[\left(\mathbf{S}^{(m)}\mathbf{W}^{(m)}\right)^\top, \left(\mathbf{P}^{(m)\top}\mathbf{D}^{(m)}\mathbf{W}^{(m)}\right)^\top\right]\left[\boldsymbol{\beta}^{(m)}, \tilde{\boldsymbol{\beta}}^{(m+1)}\right] \\
&= -\boldsymbol{a}^{(m-1)} - \mathbf{P}^{(m-1)}\tilde{\boldsymbol{\beta}}^{(m)}
\end{aligned}$$

Therefore, the claim Eq. 22 is proved with induction. Lastly, we prove the following claim where $\mathbf{q}^{(m)}$ and $c^{(m)}$ are defined in Eq. 13 and Eq. 14.

$$-\sum_{i=m+1}^{L}\boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} + \sum_{i=m+1}^{L-1}\sum_{\substack{\mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)} \\ j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)}}} \mathbf{l}_j^{(i)}[\boldsymbol{\nu}_j^{(i)}]^+ = \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)} \tag{23}$$

This claim can be proved by applying Eq. 21 and Eq. 22.

$$\begin{aligned}
&-\sum_{i=m+1}^{L}\boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} + \sum_{i=m+1}^{L-1}\sum_{\substack{\mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)} \\ j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)}}} \mathbf{l}_j^{(i)}[\boldsymbol{\nu}_j^{(i)}]^+ \\
&= -\sum_{i=m+1}^{L}\left(\mathbf{D}^{(i)}\hat{\boldsymbol{\nu}}^{(i)} - \boldsymbol{\beta}^{(i)}\mathbf{S}^{(i)}\right)^\top\mathbf{b}^{(i)} + \sum_{i=m+2}^{L}\left(-\hat{\boldsymbol{\nu}}^{(i-1)T}\underline{\mathbf{b}}^{(i-1)}\right) \\
&= \sum_{i=m+1}^{L}\left[\left(\boldsymbol{a}^{(i)\top} + \tilde{\boldsymbol{\beta}}^{(i+1)\top}\mathbf{P}^{(i)\top}\right)\mathbf{D}^{(i)}\mathbf{b}^{(i)} + \boldsymbol{\beta}^{(i)\top}\mathbf{S}^{(i)}\mathbf{b}^{(i)}\right] \\
&\quad + \sum_{i=m+2}^{L}\left(\boldsymbol{a}^{(i-1)\top} + \tilde{\boldsymbol{\beta}}^{(i)\top}\mathbf{P}^{(i-1)\top}\right)\underline{\mathbf{b}}^{(i-1)} \\
&= \sum_{i=m+1}^{L}\tilde{\boldsymbol{\beta}}^{(i)\top}\left[\mathbf{S}^{(i)}, \mathbf{P}^{(i)\top}\mathbf{D}^{(i)}\right]\mathbf{b}^{(i)} + \sum_{i=m+2}^{L}\tilde{\boldsymbol{\beta}}^{(i)\top}\mathbf{P}^{(i-1)\top}\underline{\mathbf{b}}^{(i-1)} \\
&\quad + \sum_{i=m+1}^{L}\boldsymbol{a}^{(i)\top}\mathbf{D}^{(i)}\mathbf{b}^{(i)} + \sum_{i=m+2}^{L}\boldsymbol{a}^{(i-1)\top}\underline{\mathbf{b}}^{(i-1)} \\
&= \mathbf{q}^{(m)\top}\tilde{\boldsymbol{\beta}}^{(m+1)} + c^{(m)}
\end{aligned}$$

Finally, we apply claims Eq. 22 and Eq. 23 into the dual form solution Eq. 20 and prove the Theorem 3.2.

$$
\begin{aligned}
g(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= -\sum_{i=1}^{L} \boldsymbol{\nu}^{(i)\top} \mathbf{b}^{(i)} - \hat{\boldsymbol{\nu}}^{(0)\top} x_0 - ||\hat{\boldsymbol{\nu}}^{(0)}||_1 \cdot \epsilon + \sum_{i=1}^{L-1} \sum_{\substack{\mathbf{l}_j^{(i)} < 0 < \mathbf{u}_j^{(i)} \\ j \notin \mathcal{Z}^{+(i)} \bigcup \mathcal{Z}^{-(i)}}} \mathbf{l}_j^{(i)} [\boldsymbol{\nu}_j^{(i)}]^+ \\
&= -||-\boldsymbol{a}^{(0)} - \mathbf{P}^{(0)} \tilde{\boldsymbol{\beta}}^{(1)}||_1 \cdot \epsilon + \left(\boldsymbol{a}^{(0)\top} + \tilde{\boldsymbol{\beta}}^{(1)\top} \mathbf{P}^{(0)\top} \right) x_0 + \mathbf{q}^{(0)\top} \tilde{\boldsymbol{\beta}}^{(1)} + c^{(0)} \\
&= -||\boldsymbol{a} + \mathbf{P} \tilde{\boldsymbol{\beta}}^{(1)}||_1 \cdot \epsilon + \left(\mathbf{P}^\top x_0 + \mathbf{q} \right)^\top \tilde{\boldsymbol{\beta}}^{(1)} + \boldsymbol{a}^\top x_0 + c
\end{aligned}
$$

A more intuitive proof. Here we provide another intuitive proof showing why the dual form solution of verification objective in Eq. 20 is the same as the primal one in Thereom 3.1. $d_{\text{LP}} = g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is the dual objective for Eq. 10 with free variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. We want to show that the dual problem can be viewed in the form of backward propagating $\boldsymbol{\nu}^{(L)}$ to $\hat{\boldsymbol{\nu}}^{(0)}$ following the same rules in β-CROWN. Salman et al. [30] showed that CROWN computes the same solution as the dual form in Wong and Kolter [42]: $\hat{\boldsymbol{\nu}}^{(i)}$ is corresponding to $-\mathbf{A}^{(i)}$ in CROWN (defined in the same way as in Eq. 10 but with $\boldsymbol{\beta}^{(i+1)} = 0$) and $\boldsymbol{\nu}^{(i)}$ is corresponding to $-\mathbf{A}^{(i+1)}\mathbf{D}^{(i+1)}$. When the split constraints are introduced, extra terms for the dual variable $\boldsymbol{\beta}$ modify $\boldsymbol{\nu}^{(i)}$ (highlighted in blue in Eq. 20). The way β-CROWN modifies $\mathbf{A}^{(i+1)}\mathbf{D}^{(i+1)}$ is exactly the same as the way $\boldsymbol{\beta}^{(i)}$ affects $\boldsymbol{\nu}^{(i)}$: when we split $z_j^{(i)} \geq 0$, we add $\boldsymbol{\beta}_j^{(i)}$ to the $\boldsymbol{\nu}_j^{(i)}$ in Wong and Kolter [42]; when we split $z_j^{(i)} \geq 0$, we add $-\boldsymbol{\beta}_j^{(i)}$ to the $\boldsymbol{\nu}_j^{(i)}$ in Wong and Kolter [42] ($\boldsymbol{\nu}_j^{(i)}$ is 0 in this case because it is set to be inactive). To make this relationship more clear, we define a new variable $\boldsymbol{\nu}'$, and rewrite relevant terms involving $\boldsymbol{\nu}$, $\hat{\boldsymbol{\nu}}$ below:

$$
\begin{aligned}
&\boldsymbol{\nu}_j^{(i)} = 0, \quad j \in \mathcal{Z}^{-(i)}; \\
&\boldsymbol{\nu}_j^{(i)} = \hat{\boldsymbol{\nu}}_j^{(i)}, \quad j \in \mathcal{Z}^{+(i)}; \\
&\boldsymbol{\nu}_j^{(i)} \text{ is defined in the same way as in Eq. 20 for other cases} \\
&\boldsymbol{\nu}_j^{(i)\prime} = -\boldsymbol{\beta}_j^{(i)} + \boldsymbol{\nu}_j^{(i)}, \quad j \in \mathcal{Z}^{-(i)}; \\
&\boldsymbol{\nu}_j^{(i)\prime} = \boldsymbol{\beta}_j^{(i)} + \boldsymbol{\nu}_j^{(i)}, \quad j \in \mathcal{Z}^{+(i)}; \\
&\boldsymbol{\nu}_j^{(i)\prime} = \boldsymbol{\nu}_j^{(i)}, \quad \text{otherwise} \\
&\hat{\boldsymbol{\nu}}^{(i-1)} = \mathbf{W}^{(i)\top} \boldsymbol{\nu}^{(i)\prime};
\end{aligned}
\tag{24}
$$

It is clear that $\boldsymbol{\nu}'$ corresponds to the term $-(\mathbf{A}^{(i+1)}\mathbf{D}^{(i+1)} + \boldsymbol{\beta}^{(i+1)\top}\mathbf{S}^{(i+1)})$ in Eq. 10, by noting that $\boldsymbol{\nu}^{(i)}$ in [42] is equivalent to $-\mathbf{A}^{(i+1)}\mathbf{D}^{(i+1)}$ in CROWN and the choice of signs in $\mathbf{S}^{(i+1)}$ reflects neuron split constraints. Thus, the dual formulation will produce the same results as Eq. 18, and thus also equivalent to Eq. 3.

□

Corollary 3.2.1. *When $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are optimally set, β-CROWN produces the same solution as LP with split constraints when intermediate bounds $\mathbf{l}, \mathbf{u}$ are fixed. Formally,*

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

where p^*_{LP} is the optimal objective of Eq. 10.

Proof. Given fixed intermediate layer bounds $\mathbf{l}$ and $\mathbf{u}$, the dual form of the verification problem in Eq. 10 is a linear programming problem with dual variables defined in Eq. 19. Suppose we use an LP solver to obtain the optimal dual solution $\boldsymbol{\nu}^*, \boldsymbol{\xi}^*, \boldsymbol{\mu}^*, \boldsymbol{\gamma}^*, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*$. Then we can set $\boldsymbol{\alpha}_j^{(i)} = \frac{\boldsymbol{\gamma}_j^{(i)*}}{\boldsymbol{\mu}_j^{(i)*} + \boldsymbol{\gamma}_j^{(i)*}}$, $\boldsymbol{\beta} = \boldsymbol{\beta}^*$ and plug them into Eq. 20 to get the optimal dual solution d^*_{LP}. Theorem 3.2 shows that, β-CROWN can compute the same objective d^*_{LP} given the same $\boldsymbol{\alpha}_j^{(i)} = \frac{\boldsymbol{\gamma}_j^{(i)*}}{\boldsymbol{\mu}_j^{(i)*} + \boldsymbol{\gamma}_j^{(i)*}}$, $\boldsymbol{\beta} = \boldsymbol{\beta}^*$,

thus $\max_{0\leq\boldsymbol{\alpha}\leq 1,\boldsymbol{\beta}\geq 0} g(\boldsymbol{\alpha},\boldsymbol{\beta}) \geq d^*_{\text{LP}}$. On the other hand, for any setting of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, β-CROWN produces the same solution $g(\boldsymbol{\alpha},\boldsymbol{\beta})$ as the rewritten dual LP in Eq. 20, so $g(\boldsymbol{\alpha},\boldsymbol{\beta}) \leq d^*_{\text{LP}}$. Thus, we have $\max_{0\leq\boldsymbol{\alpha}\leq 1,\boldsymbol{\beta}\geq 0} g(\boldsymbol{\alpha},\boldsymbol{\beta}) = d^*_{\text{LP}}$. Finally, due to the strong duality in linear programming, $p^*_{\text{LP}} = d^*_{\text{LP}} = \max_{0\leq\boldsymbol{\alpha}\leq 1,\boldsymbol{\beta}\geq 0} g(\boldsymbol{\alpha},\boldsymbol{\beta})$. □

The variables $\boldsymbol{\alpha}$ in β-CROWN can be translated to dual variables in LP as well. Given $\boldsymbol{\alpha}$ in β-CROWN, we can get the corresponding dual LP variables $\boldsymbol{\mu},\boldsymbol{\gamma}$ given $\boldsymbol{\alpha}$ by setting $\boldsymbol{\mu}_j^{(i)} = (1 - \boldsymbol{\alpha}_j^{(i)})[\hat{\boldsymbol{\nu}}_j^{(i)}]^-$ and $\boldsymbol{\gamma}_j^{(i)} = \boldsymbol{\alpha}_j^{(i)}[\hat{\boldsymbol{\nu}}_j^{(i)}]^-$.

A.3 Proof for soundness and completeness

Theorem 3.3. *β-CROWN with branch and bound on splitting ReLU neurons is sound and complete.*

Proof. **Soundness.** Branch and bound (BaB) with β-CROWN is sound because for each subdomain $\mathcal{C}_i := \{x \in \mathcal{C}, z \in \mathcal{Z}_i\}$, we apply Theorem 3.1 to obtain a sound lower bound $\underline{f}_{\mathcal{C}_i}$ (the bound is valid for any $\boldsymbol{\beta} \geq 0$). The final bound returned by BaB is $\min_i \underline{f}_{\mathcal{C}_i}$ which represents the worst case over all subdomains, and is a sound lower bound for $x \in \mathcal{C} := \cup_i \mathcal{C}_i$.

Completeness. To show completeness, we need to solve Eq. 1 to its global minimum. When there are N unstable neurons, we have up to 2^N subdomains, and in each subdomain we have all unstable ReLU neurons split into one of the $z_j^{(i)} \geq 0$ or $z_j^{(i)} < 0$ case. The final solution obtained by BaB is the min over these 2^N subdomains. To obtain the global minimum, we must ensure that in every of these 2^N subdomain we can solve Eq. 10 exactly.

When all unstable neurons are split in a subdomain $\mathcal{C}_i$, the network becomes a linear network and neuron split constraints become linear constraints w.r.t. inputs. Under this case, an LP with Eq. 10 can solve the verification problem in $\mathcal{C}_i$ exactly. In β-CROWN, we solve the subdomain using the usually non-concave formulation Eq. 12; however, in this case, it becomes concave in $\hat{\boldsymbol{\beta}}$ because no intermediate layer bounds are used (no $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$) and no ReLU neuron is relaxed (no $\boldsymbol{\alpha}$), thus the only optimizable variable is $\boldsymbol{\beta}$ (Eq. 12 becomes Eq. 8). Eq. 8 is concave in $\boldsymbol{\beta}$ so (super)gradient ascent guarantees to converge to the global optimal $\boldsymbol{\beta}^*$. To ensure convergence without relying on a preset learning rate, a line search can be performed in this case. Then, according to Corollary 3.2.1, this optimal $\boldsymbol{\beta}^*$ corresponds to the optimal dual variable for the LP in Eq. 10 and the objective is a global minimum of Eq. 10. □

B More details on β-CROWN with branch and bound (BaB)

B.1 β-CROWN with branch and bound for complete verification

We list our β-CROWN with branch and bound based complete verifier (β-CROWN BAB) in Algorithm 1. The algorithm takes a target NN function f and a domain $\mathcal{C}$ as inputs. The subprocedure `optimized_beta_CROWN` optimizes $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ (free variables for computing intermediate layer bounds and last layer bounds) as Eq. 12 in Section 3.3. It operates in a batch and returns the lower and upper bounds for n selected subdomains simultaneously: a lower bound is obtained by optimizing Eq. 12 using β-CROWN and an upper bound can be the network prediction $f(x^*)$ given the x^* that minimizes Eq. 7[1]. Initially, we don't have any splits, so we only need to optimize $\hat{\boldsymbol{\alpha}}$ to obtain $\underline{f}$ for $x \in \mathcal{C}$ (Line 2). Then we utilize the power of GPUs to split in parallel and maintain a global set $\mathbb{P}$ storing all the sub-domains which does not satisfy $\underline{f}_{\mathcal{C}_i} < 0$ (Line 5-10). Specifically, `batch_pick_out` extends branching strategy BaBSR [8] or FSB [11] in a parallel manner to select n (batch size) sub-domains in $\mathbb{P}$ and determine the corresponding ReLU neuron to split for each of them. If the length of $\mathbb{P}$ is less than n, then we reduce n to the length of $\mathbb{P}$. `batch_split` splits each selected $\mathcal{C}_i$ to two sub-domains $\mathcal{C}_i^l$ and $\mathcal{C}_i^u$ by forcing the selected unstable ReLU neuron to be positive and negative, respectively. `Domain_Filter` filters out verified sub-domains (proved with $\underline{f}_{\mathcal{C}_i} \geq 0$) and we insert the remaining

[1] We want an upper bound of the objective in Eq. 1. Since Eq. 1 is an minimization problem, any feasible x produces an upper bound of the optimal objective. When Eq. 1 is solved exactly as f^* (such as in the case where all neurons are split), we have $f^* = \underline{f} = \overline{f}$. See also the discussions in Section I.1 of De Palma et al. [10].

Algorithm 1 β-CROWN with branch and bound for complete verification. Comments are in brown.

1: **Inputs**: $f, \mathcal{C}, n$ (batch size), δ (tolerance), η (maximum length of sub-domains)
2: $(\underline{f}, \overline{f}) \leftarrow$ `optimized_beta_CROWN`$(f, [\mathcal{C}])$ ▷ Initially there is no split, so optimization is done over $\hat{\boldsymbol{\alpha}}$
3: $\mathbb{P} \leftarrow [(\underline{f}, \overline{f}, \mathcal{C})]$ ▷ $\mathbb{P}$ is the set of all unverified sub-domains
4: **while** $\underline{f} < 0$ **and** $\overline{f} \geq 0$ **and** $\overline{f} - \underline{f} > \delta$ **and** `length`$(\mathbb{P}) < \eta$ **do**
5: $\quad(\mathcal{C}_1, \dots, \mathcal{C}_n) \leftarrow$ `batch_pick_out`$(\mathbb{P}, n)$ ▷ Pick sub-domains to split and removed them from $\mathbb{P}$
6: $\quad[\mathcal{C}_1^l, \mathcal{C}_1^u, \dots, \mathcal{C}_n^l, \mathcal{C}_n^u] \leftarrow$ `batch_split`$(\mathcal{C}_1, \dots, \mathcal{C}_n)$ ▷ Each $\mathcal{C}_i$ splits into two sub-domains $\mathcal{C}_i^l$ and $\mathcal{C}_i^u$
7: $\quad\left[\underline{f}_{\mathcal{C}_1^l}, \overline{f}_{\mathcal{C}_1^l}, \underline{f}_{\mathcal{C}_1^u}, \overline{f}_{\mathcal{C}_1^u}, \dots, \underline{f}_{\mathcal{C}_n^l}, \overline{f}_{\mathcal{C}_n^l}, \underline{f}_{\mathcal{C}_n^u}, \overline{f}_{\mathcal{C}_n^u}\right] \leftarrow$ `optimized_beta_CROWN`$(f, [\mathcal{C}_1^l, \mathcal{C}_1^u, \dots, \mathcal{C}_n^l, \mathcal{C}_n^u])$
▷ Compute lower and upper bounds by optimizing $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ mentioned in Section 3.3 in a batch
8: $\quad\mathbb{P} \leftarrow \mathbb{P} \bigcup$ `Domain_Filter` $\left([\underline{f}_{\mathcal{C}_1^l}, \overline{f}_{\mathcal{C}_1^l}, \mathcal{C}_1^l], [\underline{f}_{\mathcal{C}_1^u}, \overline{f}_{\mathcal{C}_1^u}, \mathcal{C}_1^u], \dots, [\underline{f}_{\mathcal{C}_n^l}, \overline{f}_{\mathcal{C}_n^1}, \mathcal{C}_n^l], [\underline{f}_{\mathcal{C}_n^u}, \overline{f}_{\mathcal{C}_n^u}, \mathcal{C}_n^u]\right)$ ▷
Filter out verified sub-domains, insert the left domains back to $\mathbb{P}$
9: $\quad\underline{f} \leftarrow \min\{\underline{f}_{\mathcal{C}_i} \mid (\underline{f}_{\mathcal{C}_i}, \overline{f}_{\mathcal{C}_i}, \mathcal{C}_i) \in \mathbb{P}\}, i = 1, \dots, n$ ▷ To ease notation, $\mathcal{C}_i$ here indicates either $\mathcal{C}_i^u$ or $\mathcal{C}_i^l$
10: $\quad\overline{f} \leftarrow \min\{\overline{f}_{\mathcal{C}_i} \mid (\underline{f}_{\mathcal{C}_i}, \overline{f}_{\mathcal{C}_i}, \mathcal{C}_i) \in \mathbb{P}\}, i = 1, \dots, n$
11: **Outputs:** $\underline{f}, \overline{f}$

ones to $\mathbb{P}$. The loop breaks if the property is proved ($\underline{f} \geq 0$), or a counter-example is found in any sub-domain ($\overline{f} < 0$), or the lower bound $\underline{f}$ and upper bound $\overline{f}$ are sufficiently close, or the length of sub-domains $\mathbb{P}$ reaches a desired threshold η (maximum memory limit).

Note that for models evaluated in our paper, we find that computing intermediate layer bounds in every iteration at line 7 is too costly (although it is possible and supported) so we only compute intermediate layer bounds once at line 2. At line 7, only the neuron with split constraints have their intermediate layer bounds updated, and other intermediate bounds are not recomputed. This makes the intermediate layer bounds looser but it allows us to quickly explore a large number of nodes on the branch and bound search tree and is overall beneficial for verifying most models. A similar observation was also found in De Palma et al. [10] (Section 5.1.1).

B.2 Comparisons to other GPU based complete verifiers

Bunel et al. [6] proposed to reformulate the linear programming problem in Eq. 10 through Lagrangian decomposition. Eq. 10 is decomposed layer by layer, and each layer is solved with simple closed form solutions on GPUs. A Lagrangian is used to enforce the equality between the output of a previous layer and the input of a later layer. This optimization formulation has the same power as a LP (Eq. 10) under convergence. The main drawback of this approach is that it converges relatively slowly (it typically requires hundreds of iterations to converge to a solution similar to the solution of a LP), and it also cannot easily jointly optimize intermediate layer bounds. In Table 1 (PROX BABSR) and Figure 1 (BDD+ BABSR, which refers to the same method) we can see that this approach is relatively slow and has high timeout rates compared to other GPU accelerated complete verifiers. Recently, De Palma et al. [11] proposed a better branching strategy, filtered smart branching (FSB), to further improved verification performance of [6], but the Lagrangian Decomposition based incomplete verifier and the branch and bound procedure stay the same.

De Palma et al. [10] used a tighter convex relaxation [2] than the typical LP formulation in Eq. 10 for the incomplete verifier. This tighter relaxation may contain exponentially many constraints, and De Palma et al. [10] proposed to solve the verification problem in its dual form where each constraint becomes a dual variable. A small active set of dual variables is maintained during dual optimization to ensure efficiency. This tighter relaxation allows it to outperform [6], but it also comes with extra computational costs and difficulties for an efficient implementation (e.g. a "masked" forward/backward pass is needed which requires a customised low-level convolution implementation). Additionally, De Palma et al. [10] did not optimize intermediate layer bounds jointly.

Xu et al. [45] used CROWN [46] (categorized as a linear relaxation based perturbation analysis (LiRPA) algorithm) as the incomplete solver in BaB. Since CROWN cannot encode neural split constraints, Xu et al. [45] essentially solve Eq. 10 *without* neuron split constraints ($z_j^{(i)} \geq 0, i \in \{1, \cdots, L-1\}, j \in \mathcal{Z}^{+(i)}$ and $z_j^{(i)} < 0, i \in \{1, \cdots, L-1\}, j \in \mathcal{Z}^{-(i)}$) in Eq. 10. The missing

constraints lead to looser bounds and unnecessary branches. Additionally, using CROWN as the incomplete solver leads to incompleteness - even when all unstable ReLU neurons are split, Xu et al. [45] still cannot solve Eq. 1 to a global minimum, so a LP solver has to be used to check inconsistent splits and guarantee completeness. Our β-CROWN BaB overcomes these drawbacks: we consider per-neuron split constraints in β-CROWN which reduces the number of branches and solving time (Table 1). Most importantly, β-CROWN with branch and bound is sound and complete (Theorem 3.3) and we do not rely on any LP solvers.

Another difference between Xu et al. [45] and our method is the joint optimization of intermediate layer bounds (Section 3.3). Although [45] also optimized intermediate layer bounds, they only optimize $\boldsymbol{\alpha}$ and do not have $\boldsymbol{\beta}$, and they share the same variable $\boldsymbol{\alpha}$ for all intermediate layer bounds and final bounds, with a total of $O(Ld)$ variables to optimize. Our analysis in Section 3.3 shows that there are in fact, $O(L^2d^2)$ free variables to optimize, and we share less variables as in Xu et al. [45]. This allows us to achieve tighter bounds and improve overall performance.

B.3 Detection of Infeasibility

Maximizing Eq. 8 with infeasible constraints leads to unbounded dual objective, which can be detected by checking if this optimized lower bound becomes *greater than the upper bound* (which is also maintained in BaB, see Alg.1 in Sec. B.1). For the robustness verification problem, a subdomain that has lower bound greater than 0 is dropped, which includes the unbounded case. Due to insufficient convergence, this cannot always detect infeasibility, but it *does not affect soundness*, as this infeasible subdomain only leads to worse overall lower bound in BaB. To guarantee *completeness*, we show that when all unstable neurons are split the problem is concave (see Section A.3); in this case, we can use line search to guarantee convergence when feasible, and detect infeasibility if the objective exceeds the upper bound (line search guarantees the objective can eventually exceed upper bound). In most real scenarios, the verifier either finishes or times out before all unstable neurons are split.

C Details on Experimental Setup and Results

C.1 Experimental Setup

We run our experiments on a machine with a single NVIDIA RTX 3090 GPU (24GB GPU memory), a AMD Ryzen 9 5950X CPU and 64GB memory. Our β-CROWN solver uses 1 CPU and 1 GPU only, except for the MLP models in Table 2 where 16 threads are used to compute intermediate layer bounds with Gurobi[2]. We use the Adam optimizer [21] to solve both $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ in Eq. 12 with 20 iterations. The learning rates are set as 0.1 and 0.05 for optimizing $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ respectively. We decay the learning rates with a factor of 0.98 per iteration. To maximize the benefits of parallel computing on GPU, we use batch sizes n =1024 for Base (CIFAR-10), Wide (CIFAR-10), Deep (CIFAR-10), CNN-A-Adv (MNIST) and ConvSmall (MNIST), n =2048 for ConvSmall (CIFAR-10), n =4096 for CNN-A-Adv (CIFAR-10), CNN-A-Adv-4 (CIFAR-10), CNN-A-Mix (CIFAR-10) and CNN-A-Mix-4 (CIFAR-10), n =256 for CNN-B-Adv (CIFAR-10) and CNN-B-Adv-4 (CIFAR-10), n =1024 for ConvBig (MNIST), n =10 for ConvBig (CIFAR-10), n =8 for ResNet (CIFAR-10) respectively. The CNN-A-Adv, CNN-A-Adv-4, CNN-A-Mix, CNN-A-Mix-4, CNN-B-Adv and CNN-B-Adv-4 models are obtained from the authors or [9] and are the same as the models used in their paper. We summarize the model structures in both incomplete verification and complete verification (Base, Wide and Deep) experiments in Table A1. Our code is available at `http://PaperCode.cc/BetaCROWN`.

C.2 Additional Experiments

More results on incomplete verification In this paragraph we compare our β-CROWN FSB to many other incomplete verifiers. WK [42] and CROWN [46] are simple bound propagation methods;

[2]Note that our β-CROWN verifier does not rely on MILP/LP solvers. For these very small MLP models, we find that a MILP solver can actually compute intermediate layer bounds pretty quickly and using these tighter intermediate bounds are quite helpful for β-CROWN. This also enables us to utilize both CPUs and GPUs on a machine. For all other models, intermediate layer bounds are computed through optimizing Eq. 12. Practically, MILP is not scalable beyond these very small MLP models and these small models are not the main focus of this work.

Table A1: Model structures used in our experiments. For example, Conv(1, 16, 4) stands for a conventional layer with 1 input channel, 16 output channels and a kernel size of 4×4. Linear(1568, 100) stands for a fully connected layer with 1568 input features and 100 output features. We have ReLU activation functions between two consecutive layers.

Model name	Model structure
CNN-A-Adv (MNIST)	Conv(1, 16, 4) - Conv(16, 32, 4) - Linear(1568, 100) - Linear(100, 10)
ConvSmall (MNIST)	Conv(1, 16, 4) - Conv(16, 32, 4) - Linear(800, 100) - Linear(100, 10)
ConvBig (MNIST)	Conv(1, 32, 3) - Conv(32, 32, 4) - Conv(32, 64, 3) - Conv(64, 64, 4) - Linear(3136, 512) - Linear(512, 512) - Linear(512, 10)
ConvSmall (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(1152, 100) - Linear(100, 10)
ConvBig (CIFAR-10)	Conv(3, 32, 3) - Conv(32, 32, 4) - Conv(32, 64, 3) - Conv(64, 64, 4) - Linear(4096, 512) - Linear(512, 512) - Linear(512, 10)
CNN-A-Adv/-4 (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
CNN-B-Adv/-4 (CIFAR-10)	Conv(3, 32, 5) - Conv(32, 128, 4) - Linear(8192, 250) - Linear(250, 10)
CNN-A-Mix/-4 (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
Base (CIFAR-10)	Conv(3, 8, 4) - Conv(8, 16, 4) - Linear(1024, 100) - Linear(100, 10)
Wide (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
Deep (CIFAR-10)	Conv(3, 8, 4) - Conv(8, 8, 3) - Conv(8, 8, 3) - Conv(8, 8, 4) - Linear(412, 100) - Linear(100, 10)

CROWN-OPT uses the joint optimization on intermediate layer bounds (optimizing Eq. 12 with no $\hat{\boldsymbol{\beta}}$, as done in [45]). We also include triangle relaxation based LP verifiers with intermediate layer bounds obtained from WK, CROWN and CROWN-OPT. In our experiments in Table 1, we noticed that BIGM+A.SET BABSR [10] and Fast-and-Complete [45] are also very competitive among existing state-of-the-art complete verifiers[3] - they runs fast in many cases with low timeout rates. Therefore, we also evaluate BIGM+A.SET BABSR and Fast-and-Complete with an early stop of 3 minutes for the incomplete verification setting as an extension of Section 4.2. The verified accuracy obtained from each method are reported in Table A2. BIGM+A.SET BABSR and Fast-and-Complete sometimes produce better bounds than SDP-FO, however β-CROWN FSB consistently outperforms both of them. Additionally, we found that intermediate layer bounds are important for LP verifier on some models, although even with the tightest possible CROWN-OPT bounds the verified accuracy gap between LP verifiers and ours is still large. Additionally, LP verifiers need significantly more time.

Additional results on the tightness of verification. In Figure A1, we include LP based verifiers as baselines and compare the lower bound from verification to the upper bound obtained by PGD. The LP verifiers use the triangle relaxations described in Section 2.1, with intermediate layer bounds from WK [42], CROWN [46] and CROWN with joint optimization on intermediate layer bounds (denoted as CROWN-OPT). We find that tighter intermediate layer bounds obtained by CROWN can greatly improve the performance of the LP verifier compared to those using looser ones obtained by Wong and Kolter [42]. Furthermore, using intermediate layer bounds computed by joint optimization can achieve additional improvements. However, our branch and bound with β-CROWN can significantly outperform these LP verifiers. This shows that BaB is an effective approach for incomplete verification, outperforming the bounds produced by a single LP.

Lower bound improvements over time In Figure A2, we plot lower bound values vs. time for β-CROWN BABSR and BIGM+A.SET BABSR (one of the most competitive methods in Table 1) on the CNN-A-Adv (MNIST) model. Figure A2 shows that branch and bound can indeed quickly improve the lower bound, and our β-CROWN BABSR is consistently faster than BIGM+A.SET BABSR. In contrast, SDP-FO [9], which typically requires 2 to 3 hours to converge, can only provide very loose bounds during the first 3 minutes of optimization (out of the range on these figures).

Ablation study of running time on GPUs and CPUs We conduct the same experiments as in Table 1 but run β-CROWN FSB on CPUs instead of GPUs. As shown in Table A3, our method is strong even on a single CPU, showing that the good performance does not only come from GPU acceleration; our efficient algorithm also contributes to our success. On the other hand, using GPU can boost the performance by at least 2x. Importantly, the models evaluated in this table are very small ones. Massive parallelization on GPU will lead to more significant acceleration on larger

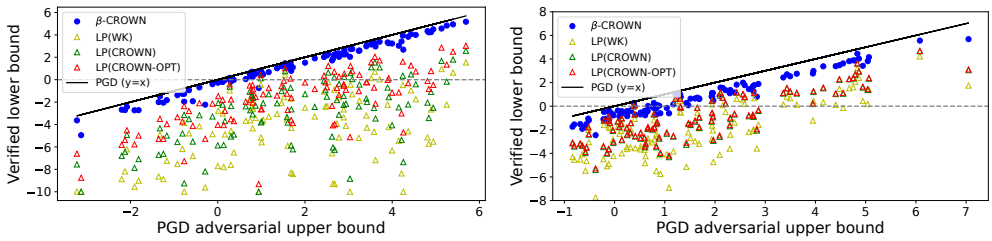
[3]The concurrent work BaDNB (BDD+ FSB) does not have public available code when our paper was submitted.

Table A2: **Verified accuracy (%)** and avg. per-example verification time (s) on 7 models from SDP-FO [9].

Dataset	MNIST $\epsilon = 0.3$		CIFAR $\epsilon = 2/255$											
Model	CNN-A-Adv		CNN-B-Adv		CNN-B-Adv4		CNN-A-Adv		CNN-A-Adv4		CNN-A-Mix		CNN-A-Mix4	
Methods	Verified%	Time (s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)
WK [42]	0	0.1	8.5	0.4	34.5	0.8	32.5	0.4	39.5	0.5	15.0	0.3	30.0	0.4
CROWN [46]	1.0	0.1	21.5	0.5	43.5	0.9	35.5	0.6	41.5	0.7	23.5	0.4	38.0	0.5
CROWN-OPT [45]	14.0	2.7	21.5	5.5	45.0	4.0	36.0	2.0	42.0	1.6	25.0	2.3	38.5	2.0
LP (WK)[§]	0.5	16	14.5	612	41.0	1361	35.0	114	41.5	140	19.0	84	36.5	117
LP (CROWN)	3.5	22	21.5	941	45.0	1570	36.0	123	41.5	147	24.0	119	38.5	126
LP (CROWN-OPT)	14.0	40	21.5	977	45.0	1451	36.0	122	42.0	152	25.0	94.8	38.5	127
SDP-FO [9][*]	43.4	>20h	32.8	>25h	46.0	>25h	39.6	>25h	40.0	>25h	39.6	>25h	47.8	>25h
PRIMA [26]	44.5	136	38.0	344	53.5	44	41.5	4.8	45.0	4.9	37.5[‡]	34	48.5	7.0
BigM+A.Set [10]	63.0	117	N/A[†]	N/A	N/A	N/A	41.0	79	**46.0**	39	30.0	122	47.0	71
Fast-and-Complete[§]	66.0	49	38.5	64	51.5	21	41.5	14	**46.0**	4.2	33.0	79	46.0	10
β-CROWN FSB	**70.5**	21	**46.5**	32	**54.0**	12	**44.0**	5.8	**46.0**	5.6	**41.5**	50	**50.5**	5.9
Upper Bound (PGD)	76.5	-	65.0	-	63.0	-	50.0	-	49.5	-	53.0	-	57.5	-

[§] Names in parentheses are methods to compute intermediate layer bounds for the LP verifier.
[*] SDP-FO results are directly from their paper due to the very long running time. All other methods are tested on the same set of 200 examples.
[†] The implementation of BigM+A.Set BaBSR is not compatible with CNN-B-Adv and CNN-B-Adv4 models which have an convolution with asymmetric padding.
[‡] A recent version (Oct 26, 2021) of [26] reported better results on CNN-A-Mix. We found that their results were produced on a selection of 100 data points, and reruning their method using the same command on the same set of 200 random examples as used in other methods in this table produces different results, as reported here.
[§] We use our β-CROWN code and turn off β optimization to emulate the algorithm used in [45]. This in fact leads to better performance than the original approach in [45] because we allow more α variables to be optimized and our implementation is generally better.

Figure A1: Verified lower bound on $f(x)$ by β-CROWN FSB compared against incomplete LP verifiers using different intermediate layer bounds obtained from [42] (denoted as LP (WK)), CROWN [46] (denoted as LP (CROWN)), and jointly optimized intermediate bounds in Eq. 12 (denoted as LP (CROWN-OPT)), v.s. the adversarial upper bound on $f(x)$ found by PGD. LPs need much longer time to solve than β-CROWN on CIFAR-10 models (see Table A2).



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

models. The speedup on multi-core CPU is not obvious, possibly due to the limitation of underlying implementations of PyTorch.

Ablation study on the impact of α, β, and their joint optimization We conduct the same experiments as in Table 1 but turn on or turn off α and β optimization to see the contribution of each part. As shown in Table A4, optimizing both α and β leads to optimal performance. Optimizing beta has a greater impact than optimizing α. Joint optimization is helpful for CIFAR10-Base and CIFAR10-Wide models, reducing the overall runtime. For simple models like CIFAR10-Deep,

Figure A2: For the CNN-A-Adv (MNIST) model, we randomly select four examples from the incomplete verification experiment and plot the lower bound v.s. time (in 180 seconds) of β-CROWN BABSR and BIGM+A.SET BABSR. Larger lower bounds are better. β-CROWN BaBSR improves bound noticeably faster in all four situations.

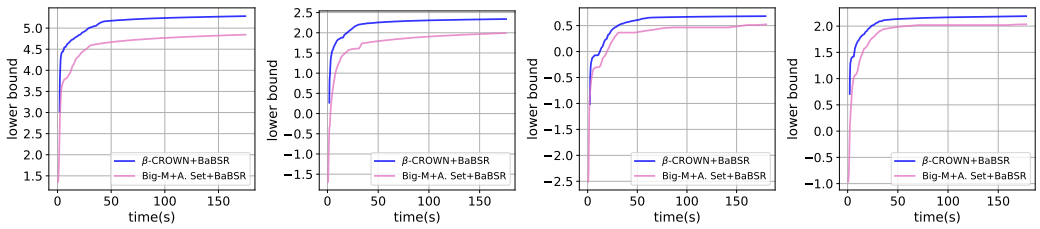


Table A3: Average runtime and average number of branches on three CIFAR-10 models over 100 properties (the same setting as in Table 1) by using different numbers of CPU cores, as well as using a single GPU.

	CIFAR-10 Base			CIFAR-10 Wide			CIFAR-10 Deep		
Hardware	time(s)	branches	%timeout	time(s)	branches	%timeout	time(s)	branches	%timeout
1 CPU	249.49	7886.37	4.00	178.01	2749.96	4.00	47.46	41.12	0.00
4 CPU	228.28	9575.52	4.00	172.55	3956.17	4.00	45.35	41.12	0.00
16 CPU	222.71	10271.08	4.00	172.40	4087.15	4.00	43.97	41.12	0.00
1 GPU	118.23	208018.21	3.00	78.32	116912.57	2.00	5.69	41.12	0.00

disabling joint optimization can help slightly because this model is very easy to verify (within a few seconds) and using looser bounds reduces verification cost.

Table A4: Ablation study on the CIFAR-10 Base, Wide and Deep models (the same setting as in Table 1), including combinations of optimizing or not optimizing α and/or β variables, and using or not using joint optimization for intermediate layer bounds.

			CIFAR-10 Base			CIFAR-10 Wide			CIFAR-10 Deep		
joint opt	α	β	time(s)	branches	%timeout	time(s)	branches	%timeout	time(s)	branches	%timeout
	✓		233.86	233233.70	6.00	148.46	113017.10	4.00	5.77	260.18	0.00
		✓	174.10	163037.05	4.00	102.65	86571.18	2.00	5.73	134.76	0.00
	✓	✓	139.83	133346.44	3.00	91.01	73713.30	2.00	5.22	100.44	0.00
✓	✓		163.69	160058.80	4.00	149.00	115509.71	4.00	8.58	65.70	0.00
✓		✓	162.95	150631.49	4.00	89.22	72479.96	2.00	8.38	52.26	0.00
✓	✓	✓	118.23	208018.21	3.00	78.32	116912.57	2.00	5.69	41.12	0.00