CHAPTER 8

Incomplete neural network verification

As mentioned in the previous section, solving the complete neural network verification problem is challenging and time-consuming, and ealier works that focused on complete verification usually do not scale to problems beyond a hundred neurons. Further, it has been shown that complete verification for ReLU networks is NP-complete (Katz et al., 2017). Motivated by the fundamental limit and practical difficulties, incomplete verification algorithms were rised since 2018, where they aim to provide sound verification for large neural networks, with the trade-off that they can only certify some but not all instances. These verifiers are often efficient and can be easily parallelized with GPU. Interestingly, these incomplete verifiers later become powerful tools even for complete verification, as will be shown in the next section.

8.1 A convex relaxation framework

We first introduce the convex relaxation framework, which covers most of the existing incomplete neural network verification methods. As mentioned in the last chapter, neural network verification can be formulated as an optimization problem. For simplicity, let us assume that $f(\boldsymbol{x})$ is an L-layer feedforward network:

$$f(\boldsymbol{x}) := \boldsymbol{z}^{(L)} = W^{(L)}\boldsymbol{x}^{(L)} + \boldsymbol{b}^{(L)}, \quad \boldsymbol{x}^{(l)} = \sigma^{(l)}(W^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}), \quad l \in [L], \tag{8.1}$$

where $\boldsymbol{x}^{(0)} = \boldsymbol{x}$ is the input, $W^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias vector of the lth layer, and $\sigma^{(l)}$ is a (nonlinear) activation function. Instead of passing a single $\boldsymbol{x}$ to the network, neural network verification aims to propagate an input region S and verify some properties of output neurons. This can be written as the following optimization problem:

$$\min_{\boldsymbol{x}^{(0)} \in S} \ \boldsymbol{c}^T \boldsymbol{x}^{(L)} \ \text{s.t.} \ \begin{aligned} &\boldsymbol{z}^{(l)} = W^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}, \\ &\boldsymbol{x}^{(l+1)} = \sigma^{(l)}(\boldsymbol{z}^{(l)}), \\ &l \in [L], \end{aligned} \tag{$\mathcal{O}$}$$

Adversarial Robustness for Machine Learning
https://doi.org/10.1016/B978-0-12-824020-5.00018-1

where $\boldsymbol{c}$ corresponds to the specification as defined in (7.8). Unfortunately, solving ($\mathcal{O}$) is intractable due to the nonconvex constraints; it is thus natural to conduct convex relaxation on the feasible set. Specifically, we relax each $x^{(l+1)} = \sigma^{(l)}(z^{(l)})$ by convex inequality constraints, leading to the following convex relaxation:

$$\begin{aligned} \min_{\boldsymbol{x}^{(0)} \in S} \boldsymbol{c}^T \boldsymbol{x}^{(L)} \quad \text{s.t. } & \boldsymbol{z}^{(l)} = W^{(l)} \boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}, \\ & \underline{\sigma}^{(l)}(\boldsymbol{z}^{(l)}) \le \boldsymbol{x}^{(l+1)} \le \bar{\sigma}^{(l)}(\boldsymbol{z}^{(l)}), \\ & \underline{\boldsymbol{z}}^{(l)} \le \boldsymbol{x}^{(l)} \le \bar{\boldsymbol{z}}_l, \\ & l \in [L], \end{aligned} \tag{$\mathcal{C}$}$$

where $\underline{\sigma}^{(l)}(\boldsymbol{z})$ ($\bar{\sigma}^{(l)}(\boldsymbol{z})$) is convex (concave) and satisfies $\underline{\sigma}^{(l)}(\boldsymbol{z}) \le \sigma^{(l)}(\boldsymbol{z}) \le \bar{\sigma}^{(l)}(\boldsymbol{z})$ for $\underline{\boldsymbol{z}}^{(l)} \le \boldsymbol{z} \le \bar{\boldsymbol{z}}^{(l)}$. The preactivation upper and lower bounds $\bar{\boldsymbol{z}}^{(l)}$ and $\underline{\boldsymbol{z}}^{(l)}$ correspond to some precomputed upper and lower bounds for each $\boldsymbol{z}$. In general, we can set $\underline{\boldsymbol{z}}^{(l)} = -\infty$ and $\bar{\boldsymbol{z}}^{(l)} = \infty$ for all l, but tighter ranges of $\underline{\boldsymbol{z}}^{(l)}$ and $\bar{\boldsymbol{z}}^{(l)}$ can achieve tighter solutions. A typical approach is to solve ($\mathcal{C}$) recursively to obtain the tightest $\underline{\boldsymbol{z}}^{(l)}$ and $\bar{\boldsymbol{z}}^{(l)}$.

In the following, we will introduce a family of linear relaxation-based methods, which can be viewed as particular cases of ($\mathcal{C}$) when the nonlinear activation functions are bounded by linear upper and lower bounds.

8.2 Linear bound propagation methods

Linear bound propagation is a simple yet effective method for sound but incomplete verification. Note that there were several papers proposing different versions of linear bound propagation methods, such as (Wong and Kolter, 2017; Weng et al., 2018a; Zhang et al., 2018; Singh et al., 2018a, 2019b) (see Fig. 8.1). However, since they lead to similar algorithms with slightly different choices of upper/lower bounds, we will mainly focus on introducing the CROWN algorithm (Zhang et al., 2018) in this section and then talk about the relationship between CROWN and other algorithms.

The key idea is to use linear functions to bound the values of each particular neuron in any intermediate and final layer. The main difficulty for obtaining a linear bound is the nonlinear activations. If there are no nonlinear activations, then clearly any $z^{(l)}$ can be written as a linear function to the input. Then for any output neuron $f(x)$, we can write it as $f(x) = a^T x$, where a is the slope, and in this case, verification is equivalent to

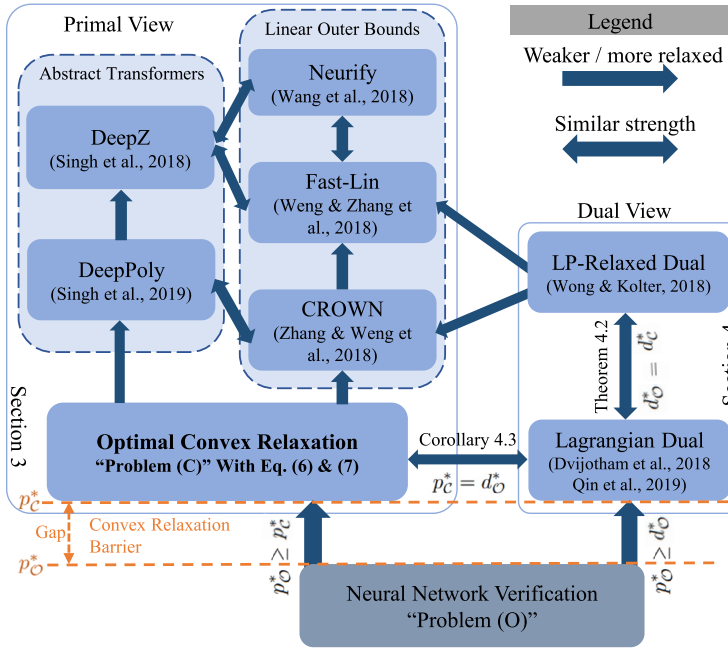

Figure 8.1 Illustration of the convex relaxation methods and convex relaxation barrier from (Salman et al., 2019b), where they showed almost all the incomplete verification algorithms can be formulated based on a convex relaxation framework and summarized the relationships between different incomplete verifiers.

finding

$$\max_{x' \in S} a^T x'.$$

For example, when S is an ℓ_∞ ball with radius ϵ centered at x_0, the worst-case output will be $\epsilon \|a\|_1 + a^T x_0$.

What if we have some nonlinear activations such as ReLU? The main idea of linear bound propagation methods is forming linear upper and lower bounds $\bar{\sigma}$ and $\underline{\sigma}$ for the activation function. Consider a ReLU function, and for simplicity, we assume the current preactivation lower and upper bounds $z \in [l, u]$, and we aim to bound $\mathrm{ReLU}(z) = \max(z, 0)$ by linear functions. We can consider three cases illustrated in Fig. 8.2. If $l \geq 0$, then we know that z is always positive, and thus $\mathrm{ReLU}(z) = z$ will be linear. If $u \leq 0$, then we know that z is always negative, and thus $\mathrm{ReLU}(z) = 0$ will also be linear. The third case $l < 0 < u$ is the most tricky, where ReLU is truly nonlinear, and we need to find linear upper and lower bounds. We call these "unstable neurons".

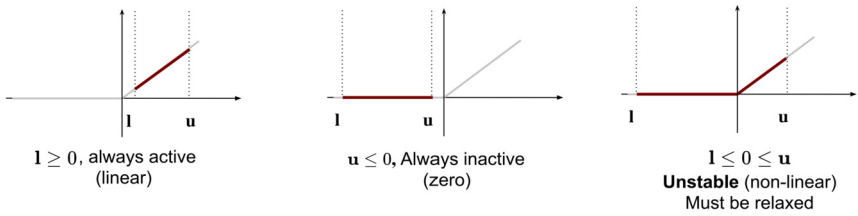

Figure 8.2 Illustration of the three cases of the ReLU activations.

For these unstable neurons, we can derive the following linear bounds:

$$\underline{\sigma}^{(l)}(z) := \alpha^{(l)} z, \quad \overline{\sigma}^{(l)}(z) := \frac{\overline{z}^{(l)}}{\overline{z}^{(l)} - \underline{z}^{(l)}}(z - \underline{z}^{(l)}) \tag{8.2}$$

with some $\alpha^{(l)} \in [0, 1]$. The upper and lower bounds are illustrated in Fig. 8.3. We can easily verify that for any $\alpha^{(l)} \in [0, 1]$, the output value of ReLU activation will be bounded by these two functions:

$$\underline{\sigma}^{(l)}(z) \leq \max(z, 0) \leq \bar{\sigma}^{(l)}(z) \quad \forall z \in [\underline{z}^{(l)}, \bar{z}^{(l)}].$$

Interestingly, there could be multiple choices for the lower bounds. Some common choices are: using the same slope of upper and lower bounds; using 0 slope for the lower bound; using some heuristics to adaptively choose the slope. More recently, it has been shown that one can choose the lower bound to optimize the final verification performance, as will be discussed in the end of this section.

With the linear upper and lower bounds for the ReLU function, we can then bound the output of a ReLU layer, where the constants are determined by slope of the linear functions used in (8.2):

Lemma 1 (ReLU relaxation in CROWN). *Given $w, v \in \mathbb{R}^d$, $\boldsymbol{u} \leq v \leq \boldsymbol{l}$ (elementwise), we have*

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

where $\mathcal{D}$ is a diagonal matrix containing free variables only when $\boldsymbol{u}_j > 0 > \boldsymbol{l}_j$ and $w_j \geq 0$, whereas its remaining values and constant b' are determined by $\boldsymbol{l}$, $\boldsymbol{u}$, w.

With linear upper and lower bounds for each neuron, we can linearize the whole neural network step by step, where at each step, we form linear upper and lower bounds for neurons at layer l. This results in an efficient

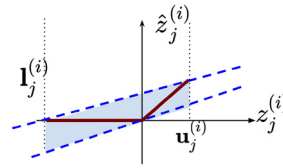

Figure 8.3 Illustration of forming a linear upper and lower bound of an unstable neuron.

back-substitution procedure to derive a linear lower bound of NN output with respect to x.

Lemma 2 (Linear Relaxation Bound). *Given an L-layer ReLU NN* $f(x) : \mathbb{R}^{d_0} \to \mathbb{R}$ *with weights* $\mathcal{W}_i$*, biases* $\mathbf{b}^i$*, and input constraint* $x \in S$*, we have*

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

where $\boldsymbol{a}_{\text{CROWN}}$ *and* c_{CROWN} *can be computed using* $\mathcal{W}_i$*,* $\mathbf{b}^i$*,* $\boldsymbol{l}_i$*,* $\boldsymbol{u}_i$ *in polynomial time. The detailed procedure can be found in several papers such as (Weng et al., 2018a; Zhang et al., 2018; Singh et al., 2019b).*

When S 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 (Xu et al., 2020b) and TPUs (Zhang et al., 2020a), and it can be a few magnitudes faster than a linear programming (LP) verifier, which is hard to parallelize on GPUs. CROWN was generalized to general architectures, whereas we only demonstrate it for feedforward ReLU networks for simplicity. Additionally, Xu et al. (2021) showed that it is possible to optimize the slope of the lower bound α using gradient ascent to further tighten the bound (sometimes referred to as α-CROWN).

Fast-Lin (Weng et al., 2018a) and Neurify (Wang et al., 2018a) assigned $\alpha^{(l)} = \frac{\overline{z}^{(l)}}{\overline{z}^{(l)} - \underline{z}^{(l)}}$, which is strictly looser than Crown but enjoys slight computational benefit. DeepZ (Singh et al., 2018a) and DeepPoly (Singh et al., 2019b) are discovered from the formal verification community using the abstract interpretation approach but can also be covered in this framework.

The optimal layerwise convex relaxation

Although the discussions have been focused on the robustness verification of ReLU networks, the above-mentioned linear bound propagation

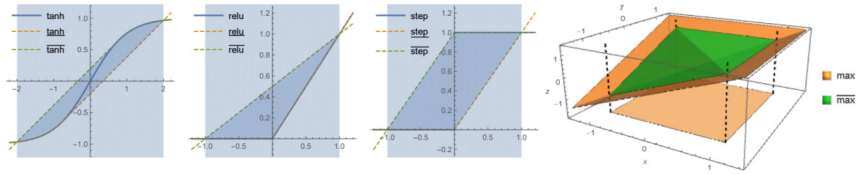

Figure 8.4 Optimal convex relaxations for common nonlinearities. For tanh, the relaxation contains two linear segments and parts of the tanh function. For ReLU and the step function, the optimal relaxations are written as 3 and 4 linear constraints, respectively. For $z = \max(x, y)$, the light orange (light gray in print version) shadow indicates the preactivation bounds for x and y, and the optimal convex relaxation is lower bounded by the max function itself. More details can be found in (Salman et al., 2019b).

methods can be easily extended to any network with general activation functions. The only requirement is to form a linear upper and lower bound for the activation function given the pre-activation bounds. More details can be found in (Zhang et al., 2018).

If we go beyond linear upper and lower bounds and use general convex functions instead, we can design the optimal convex relaxation $\underline{\sigma}$ and $\bar{\sigma}$ by the following convex envelop within $[\underline{z}, \bar{z}]$. For instance, the optimal lower bound is

$$\underline{\sigma}_{\text{opt}}(z) := \sup_{(\alpha,\gamma)\in\mathcal{A}} \alpha^\top z + \gamma, \ \mathcal{A} = \{(\alpha, \gamma) : \alpha^\top z' + \gamma \le \sigma(z'), \forall z' \in [\underline{z}, \bar{z}]\}.$$

This gives the tightest bound achieved by convex relaxation framework *for general activation functions* as shown in Fig. 8.4. With the optimal convex relaxation, one can get the tightest result but with significantly increased time complexity. For instance, for ReLU activation, Fig. 8.4 showed that the optimal convex relaxation is one linaer upper bound and the union of two linear lower bounds. Although this can still be expressed in a simple form, from the bound propagation perspective this will lead to exponentially growing number of linear bounds, which has to be solved by a linear programming solver. Therefore, in practice linear bound propagation methods are still the most popular choice for incomplete verification. It has been also discussed in (Salman et al., 2019b) regarding the gap between the optimal convex relaxation versus linear bound propagation. For some ReLU networks, they form two linear lower bounds for each ReLU activation and solve the resulting problem by linear programming. They demonstrated that the gap between the optimal convex relaxation and linear bound propagation could be reasonable in some realistic cases, but there is still some gap between the optimal convex relaxation and the exact verification problem,

also known as the "*convex relaxation barrier*", as illustrated in Fig. 8.4 (the orange (light gray in print version) gap between ($\mathcal{O}$) and ($\mathcal{C}$)).

8.3 Convex relaxation in the dual space

Another way to solve the robustness verification problem ($\mathcal{C}$) is to solve its dual problem. Its Lagrangian dual can be written as

$$g_{\mathcal{C}}(\mu^{[L]}, \underline{\lambda}^{[L]}, \overline{\lambda}^{[L]}) := \min_{x^{[L+1]}, z^{[L]}} \quad c^\top x^{(L)} + \sum_{l=0}^{L-1} \mu^{(l)\top}(z^{(l)} - W^{(l)} x^{(l)} - \mathbf{b}^{(l)}) - \sum_{l=0}^{L-1} \underline{\lambda}^{(l)\top}(x^{(l+1)} - \underline{\sigma}^{(l)}(z^{(l)})) + \sum_{l=0}^{L-1} \overline{\lambda}^{(l)\top}(x^{(l+1)} - \overline{\sigma}^{(l)}(z^{(l)})). \tag{8.3}$$

By weak duality

$$d^*_{\mathcal{C}} := \max_{\mu^{[L]}, \underline{\lambda}^{[L]} \geq 0, \overline{\lambda}^{[L]} \geq 0} g_{\mathcal{C}}(\mu^{[L]}, \underline{\lambda}^{[L]}, \overline{\lambda}^{[L]}) \leq p^*_{\mathcal{C}}, \tag{8.4}$$

but in fact Salman et al. (2019b) showed strong duality under mild conditions, which means the equality can hold in most cases.

Based on this primal-dual relationship, another way to find the lower bound of the primal objective is to find a dual solution. When the relaxed bounds $\underline{\sigma}$ and $\overline{\sigma}$ are linear, the dual objective (8.4) can be further simplified. Wong and Kolter (2018) proposed to solve this dual form with the following choices of the linear relaxation functions:

$$\underline{\sigma}^{(l)}(\boldsymbol{z}(l)) := \alpha^{(l)} \boldsymbol{z}(l), \quad \overline{\sigma}^{(l)}(\boldsymbol{z}(l)) := \frac{\overline{z}^{(l)}}{\overline{z}^{(l)} - \underline{z}^{(l)}}(\boldsymbol{z}(l) - \underline{z}^{(l)}),$$

and $0 \leq \alpha^{(l)} \leq 1$ represents the slope of the lower bound. This is equivalent to the choices of linear relaxations introduced in Fig. 8.3. When $\alpha^{(l)} = \frac{\overline{z}^{(l)}}{\overline{z}^{(l)} - \underline{z}^{(l)}}$, the greedy algorithm also recovers Fast-Lin (Weng et al., 2018a), which explains the arrow from Wong and Kolter (2018) to Weng et al. (2018a) in Fig. 8.1. Several other relationships between dual algorithms and primal algorithms can be analyzed similarly, as discussed in (Salman et al., 2019b).

8.4 Recent progresses in linear relaxation-based methods

We briefly discuss several other improvements on the above mentioned linear relaxation-based incomplete verification methods:

- Adaptive linear bound selections: As shown in the previous sections, the linear relaxation bound for each neuron in (8.2) is controlled by α, and some heuristic ways for choosing α have been discussed in CROWN. Interestingly, it is possible to automatically obtain an optimized α by optimizing the bound tightness. More specifically, for a set of given α, the final verification can be presented in Lemma 2, and we can thus try to formulate α in a^T_{CROWN} as learnable parameters and optimize for α to get tighter bounds. More details can be found in (Xu et al., 2021).
- Breaking the convex relaxation barrier: The framework mentioned above assumes convex upper and lower bounds for each single neuron. However, it has been observed that forming a convex relaxation for a group of neurons leads to tighter results, then relaxing each neuron independently. However, how to group more relevant neurons is still an open research topic. See more discussions in (Tjandraatmadja et al., 2020).
- Automatic bound propagation: Existing verification methods often focus on feed-forward ReLU networks. Only few work tried to extend existing verification methods to other models: Ko et al. (2019) extended Crown to recurrent neural networks; and Shi et al. (2020) extended Crown to the Transformer architecture. Unfortunately, each of these requires re-deriving and implementing the verification algorithm for each network architecture, which is hard even for a senior machine learning researcher. Instead of extending the proposed algorithm to each network architecture one-by-one, Xu et al. (2020a) developed an **automatic** verification framework where the mechanism is similar to auto-differentiation for back-propagation in Tensorflow/Pytorch. Users only need to write the model in Pytorch or Tensorflow, and the tool will automatically conduct verification and output the measurement without any additional human effort. This is done by representing the target function f as a given computational graph, where each edge corresponds to a *basic operation* such as addition, multiplication, ReLU, softmax, sigmoid. Nodes in this graph can be divided into input nodes, output nodes, intermediate nodes and constant nodes, and the graph is supposed to be a Directed Acyclic Graph (DAG). Linear bound propagation algorithms mentioned in this section can then be easily generalized to the computational graph – each relaxation is done on an edge (operation) instead of a neural network layer. With this framework, verification can be automatically done for any given computational graph, as long as we have the propagation rule for basic

operations. The overall usage is similar to how gradient is computed by auto-differentiation.

8.5 Extended reading

- We refer the readers to (Salman et al., 2019b) on detailed discussions about various incomplete verification algorithms.
- For linear relaxation-based methods, we encourage the readers to read (Wong and Kolter, 2017), who focused on using those bounds for certified robust training.
- Beyond verification for input perturbations, similar methodology can be applied to verifying weight perturbations in neural networks (Weng et al., 2020).
- A probabilistic framework for robustness certification based on additive noises following a given distributional characterization is proposed in (Weng et al., 2019).
- A robustness certification framework optimized for convolutional neural networks (CNN-Cert) is proposed in (Boopathy et al., 2019).