DROWN: Towards Tighter LiRPA-based Robustness Certification

Yunruo Zhang[1], Tianyu Du[2], Shouling Ji[3], Shanqing Guo[1],
[1]School of Cyber Science and Technology, Shandong University,
[2]School of Software Technology, Zhejiang University,
[3]College of Computer Science and Technology, Zhejiang University,
zhangyunruo@mail.sdu.edu.cn,{zjradty,sji}@zju.edu.cn,guoshanqing@sdu.edu.cn

Abstract

The susceptibility of deep neural networks to adversarial attacks is a well-established concern. To address this problem, robustness certification is proposed, which, unfortunately, suffers from precision or scalability issues. In this paper, we present DROWN (**D**ual C**ROWN**), a novel method for certifying DNNs' robustness. The advantage of DROWN is that it tightens classic LiRPA-based methods yet maintains similar scalability, which comes from refining pre-activation bounds of ReLU (and other activations) relaxations using two pairs of linear bounds derived from different relaxations of ReLU units in previous layers. The extensive evaluations show that DROWN achieves up to 83.39% higher certified robust accuracy than the baseline on CNNs and up to 4.68 times larger certified radii than the baseline on Transformers. Meanwhile, the running time of DROWN is about twice that of the baseline.

1 Introduction

Benefiting from big data and parallel computing, deep neural networks (DNNs) have achieved great success in various applications, such as image classification (He et al., 2016; Szegedy et al., 2016) and natural language processing (Chen et al., 2017; Vaswani et al., 2017; Lei et al., 2018). Despite their widespread use, extensive researches (Szegedy et al., 2014; Goodfellow et al., 2015; Carlini and Wagner, 2017; Madry et al., 2018) have revealed a vulnerability in DNNs to adversarial attacks. These attacks involve the meticulous crafting of adversarial examples, achieved by subtly modifying clean examples (e.g., replacing some words with their synonyms) to deceive the model. The existence of adversarial examples significantly hinders the application of DNN-based models in security-sensitive domains such as fraud detection and toxic content detection.

To address the challenge caused by adversarial attacks, recent researchers have been devoted to *robustness certification* (also known as certified robustness) (Zhang et al., 2018; Gehr et al., 2018; Gowal et al., 2019). This paradigm employs mathematical techniques to formally verify the presence of adversarial examples within a specified neighborhood around a clean example. Unfortunately, most methods focus on image classification and certify simple models composed of only affine transformations and element-wise activations (e.g., CNNs) (Du et al., 2021; Li et al., 2020), which are usually challenged by NLP models due to challenges including sequential inputs, feed-back architectures (Ko et al., 2019), and self-attention layers (Shi et al., 2020). In fact, only a few of them (e.g., several LiRPA-based methods including this work) can be applied to NLP models such as Transformers.

To handle the non-linearity of DNNs (e.g., ReLU units in Transformers), some robustness certification methods (Bunel et al., 2018) adopt the branch-and-bound (BaB) approach, which solve the certification problem using the divide-and-conquer strategy, i.e., they divide the original problem into sub-problems by splitting non-linear ReLUs into two linear halves. BaB-based methods are usually *complete*, meaning that they can gain exact results. However, their superior precision is at the cost of super-polynomial time since the problem of robustness certification is proved to be NP-complete (Katz et al., 2017; Weng et al., 2018). As a result, it is extremely challenging to apply BaB-based methods to large-scale models such as Transformers. To certify larger models, other methods relax non-linear functions into linear areas, including IBP, zonotopes, and polytopes (among which LiRPA-based methods, e.g., CROWN (Zhang et al., 2018), are shown to be the most promising according to the full benchmark results provided by Li et al., 2023). Relaxation-based methods trade precision for efficiency, which results in conservative certifications. Therefore, researchers have been pursuing tighter relaxations (Salman et al., 2019; Zhang

Proceedings of the 31st International Conference on Computational Linguistics, pages 6212–6229
January 19–24, 2025.

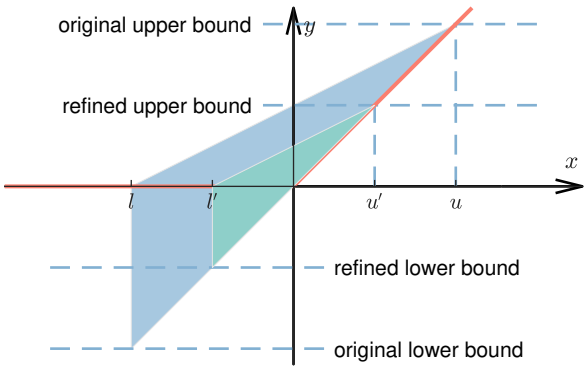


Figure 1: Refining pre-activation bounds (from (l, u) to (l', u')) to tighten ReLU ($y = \max(x, 0)$) relaxations.

et al., 2022), which is a hot topic in robustness certification currently.

There are mainly two approaches to tighten LiRPA-based robustness certification methods. The first is to optimize the linear bounds of ReLU units using gradient-based method, such as FROWN (Lyu et al., 2020) and alpha-CROWN (Xu et al., 2021). However, as the scale of the DNN grows, the number of optimization variables increases rapidly, leading to a more challenging and time-consuming optimization problem. The second is to bounding multiple ReLU units jointly instead of bounding each one of them separately, such as k-ReLU (Singh et al., 2019a) and PRIMA (Müller et al., 2022). Unfortunately, as pointed out by Tjandraatmadja et al. (2020), the tightest convex polytope may contain an exponential number of linear constraints, which leads to about $70\times \sim 1000\times$ more running time than classic LiRPA-based methods (according to Singh et al., 2019a and Müller et al., 2022). Tighter relaxations require more computation and memory in general, which hinders their application to larger models, while the rapid growth of DNNs' scale has been a trend in recent years. This stark contrast prompts us to question: *is it possible to tighten LiRPA-based methods without significantly compromising the scalability?*

In this work, we answer this question and propose a novel robustness certification method called DROWN (**D**ual C**ROWN**), based on the key insight that refining pre-activation bounds is an effective and efficient way to tighten ReLU relaxations (as shown in Fig. 1). Specifically, in contrast to classic LiRPA-based methods that propagate one pair of linear bounds through DNNs, DROWN propagates two pairs of linear bounds that use different ReLU relaxations. As a result, the pre-activation bounds of ReLU units in later layers calculated from each pair of linear bounds in DROWN are different and such difference can be used to refine the pre-activation bounds. We exhaustively evaluate DROWN to validate its performance across diverse network architectures and compare it to the current state-of-the-art methods. The evaluation results show that DROWN achieves up to 83.39% higher certified robust accuracy than the baseline on CNNs and up to 4.68 times larger certified radii than the baseline on Transformers. Meanwhile, DROWN consumes roughly twice the time of the baseline methods. These results solidify the conclusion that DROWN can improve the precision of robustness certification at an acceptable cost.

Main Contributions.

- We propose a novel robustness certification method called DROWN, which tightens LiRPA-based robustness certification while maintaining adequate scalability.
- The key design behind its advantage is an algorithm that propagates two pairs of linear bounds using different relaxations to refine pre-activation bounds of ReLU (and other activations) units in later layers of DNNs.
- Through extensive evaluations, we demonstrate that DROWN surpasses the baseline by up to 83.39% in certified robust accuracy on CNNs and achieves up to 4.68 times larger certified radii on Transformers, albeit with approximately twice the running time.

2 Related Work

Complete methods. Early complete methods usually model the robustness certification problem as a satisfiability modulo theories (SMT) (Ehlers, 2017; Katz et al., 2017; Bunel et al., 2018) or mixed integer linear programming (MILP) (Lomuscio and Maganti, 2017; Cheng et al., 2017; Tjeng et al., 2019) problem. Since the above problems lack efficient solutions, such methods typically do not scale well. Later, linear programming (LP) and BaB are composed to form the common framework of BaB-based robustness certification (Bunel et al., 2018; Lu and Kumar, 2020), which splits hidden layer activations (e.g., ReLU units) into sub-domains. Since splitting all ReLU units is time-consuming, the recent trend in BaB-based robustness certification has been finding heuristics to select good nodes to split. Nevertheless, as reported in recent

papers (Li et al., 2023), BaB-based methods can certify models (without special training) with up to 10^4 neurons and about 6 layers.

Incomplete methods. To certify the robustness of large models with general activation functions, incomplete methods employ relaxed approaches including Linear Programming (LP) (Weng et al., 2018; Salman et al., 2019; Tjandraatmadja et al., 2020), polytopes (Zhang et al., 2018; Boopathy et al., 2019; Singh et al., 2019b), zonotopes (Gehr et al., 2018; Mirman et al., 2018; Singh et al., 2018), interval bound propagation (IBP) (Gowal et al., 2019), and dual optimization (Wong and Kolter, 2018; Wong et al., 2018; Dvijotham et al., 2018). Among the above methods, LiRPA-based methods (e.g., CROWN (Zhang et al., 2018)) achieve a good balance between precision and efficiency. To further tighten LiRPA-based methods, FROWN (Lyu et al., 2020) and alpha-CROWN (Xu et al., 2021) optimize the linear bounds of hidden layer activations using gradient-based methods while k-ReLU (Singh et al., 2019a) and PRIMA (Müller et al., 2022) try to bound multiple ReLU units jointly. However, those methods trade efficiency for precision, which hurts the scalability.

3 Preliminaries

3.1 Adversarial Robustness

DNNs for Classification. Given an input sample (image or text) $x \in \mathbb{R}^{n_0}$, a classification model $f : \mathbb{R}^{n_0} \mapsto \mathbb{R}^C$ calculates numerical scores $y \in \mathbb{R}^C$ for every class. The prediction of f (i.e., F) is the label with the highest score, i.e., $F(y) = \arg\max_{c\in[C]} y_c$, where y_c is y's c-th component.

Adversarial Attack. Adversarial attackers modify a clean sample x_0 to find an adversarial example x' that triggers misclassification, i.e., $F(f(x_0)) \neq F(f(x'))$. To maintain imperceptibility, adversarial examples have to be close to the clean example, e.g., their distance is smaller than a small positive number ϵ under the ℓ_p norm in the embedding space. We refer to the set containing all potential adversarial examples as adversarial space, denoted by $\mathcal{S}(x_0)$ or $\mathcal{S}$. For image classification, $\mathcal{S}(x_0)$ is an ℓ_p ball with a radius ϵ around x_0, i.e., $\mathcal{S}(x_0) = \{x : \|x - x_0\|_p \leq \epsilon\}$. For NLP attacks by perturbing embedding vectors, $\mathcal{S}(x_0)$ is an ℓ_p ball with a radius ϵ around x_0's embedding vector. For synonym substitution attacks, $\mathcal{S}(x_0)$ is the smallest box containing all synonyms' embedding vectors.

Adversarial Robustness. A classification model f is adversarially robust within a certain adversarial space $\mathcal{S}(x_0)$ if it consistently predicts the target class $F(f(x_0))$ for all samples in $\mathcal{S}(x_0)$, i.e.,

$$F(f(x)) = F(f(x_0)), \forall x \in \mathcal{S}(x_0). \quad (1)$$

3.2 Robustness Certification

The condition of adversarial robustness, i.e., Eq. 1, is equivalent to the following condition.

$$\begin{aligned} F(y) = F(f(x_0)), \forall y \in \mathcal{R}(f, \mathcal{S}), \\ \mathcal{R}(f, \mathcal{S}) = \{y : y = f(x), \forall x \in \mathcal{S}\}. \end{aligned} \quad (2)$$

Calculating the exact $\mathcal{R}(f, \mathcal{S})$ is impractical on large DNNs due to the expensive computational cost. As a result, many robustness certification methods, such as LiRPA-based methods, calculate an over-approximation of $\mathcal{R}(f, \mathcal{S})$, i.e., $\mathcal{R}_o(f, \mathcal{S}) \supset \mathcal{R}(f, \mathcal{S})$, for better efficiency. Since a model f satisfying the following condition unquestionably satisfies Eq. 2, the results of those methods are *sound*.

$$F(y) = F(f(x_0)), \forall y \in \mathcal{R}_o(f, \mathcal{S}). \quad (3)$$

LiRPA-based methods calculate lower and upper bounds for each neuron in DNNs in a layer-by-layer way to eventually get lower and upper bounds for each score $y_c (c \in [C])$, i.e., $l_c \leq y_c \leq u_c$. Let t be the ground-truth label, f is certified to be robust within $\mathcal{S}(x_0)$ if the lower bound of ground-truth score is larger than the upper bound of any other score, i.e.,

$$l_t - \max_{c\in C, c\neq t} u_c > 0. \quad (4)$$

3.3 LiRPA Solution

In this work, we adopt LiRPA (Xu et al., 2020), a popular family of LiRPA-based methods (e.g., CROWN-BaF (Shi et al., 2020)), as the bounding procedure for robustness certification.

Linear Bounds and Propagation. Here we briefly introduce the procedure of LiRPA. Let $z_{i,j}$ be the j-th neuron in the i-th layer ($j \in [n_i], i \in [N]$). The lower and upper bound of $z_{i,j}$ with respect to neurons in the previous layer is

$$A^{i,L}_{j,:} z_{i-1} + B^{i,L}_j \leq z_{i,j} \leq A^{i,U}_{j,:} z_{i-1} + B^{i,U}_j, \quad (5)$$

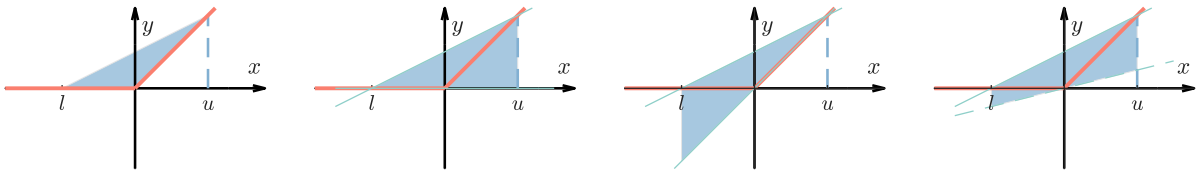


Figure 2: Different relaxations of ReLU: (a) the tightest relaxation used in LP-based methods, (b, c) two efficient relaxations used in most LiRPA-based methods, (d) the optimized relaxation with adjustable lower bound.

where $A^{i,L/U} \in \mathbb{R}^{n_i \times n_{i-1}}$ and $B^{i,L/U} \in \mathbb{R}^{n_i}$ are parameters of lower and upper bounds. The linear bounds (i.e., Eq. 5) can be propagated backward to the previous layers by substituting z_{i-1} with its linear bounds with respect to z_{i-2} as follows.

$$\begin{aligned} &A^{i,L,+}_{j,:}(A^{i-1,L}z_{i-2} + B^{i-1,L}) + \\ &A^{i,L,-}_{j,:}(A^{i-1,U}z_{i-2} + B^{i-1,U}) + B^{i,L}_j \\ &\leq z_{i,j} \leq \\ &A^{i,U,+}_{j,:}(A^{i-1,U}z_{i-2} + B^{i-1,U}) + \\ &A^{i,U,-}_{j,:}(A^{i-1,L}z_{i-2} + B^{i-1,L}) + B^{i,U}_j, \end{aligned} \tag{6}$$

where $A^{:,+}$ denotes a copy of $A^{:}$ in which the non-positive elements are set to be zeros and $A^{:,-}$ replaces non-negative elements in $A^{:}$ with zeros.

Numerical Bounds Calculation. By conducting the above substitution recursively, we can derive $z_{i,j}$'s linear bounds with respect to the input $x = z_0$ as follows.

$$A^{0,L}_{j,:}z_0 + B^{0,L}_j \leq z_{i,j} \leq A^{0,U}_{j,:}z_0 + B^{0,U}_j. \tag{7}$$

Since $z_0 = x \in \mathcal{S}(x_0)$, we can combine it with Eq. 7 to get numerical bounds of $z_{i,j}$.

Linear Bounds of Linear Layers. Since layers in DNNs can be linear or non-linear, the calculation of parameters in Eq. 5 varies according to the type of layers. Specifically, if the i-th layer is a linear layer, e.g., $z_{i,j} = \alpha^i z_{i-1} + \beta^i$, we have the following equations.

$$A^{i,L} = A^{i,U} = \alpha^i, B^{i,L} = B^{i,U} = \beta^i. \tag{8}$$

Linear Bounds of Non-linear Layers. When the i-th layer is non-linear, e.g., ReLU units, the calculation is more complicated. In this work, we relax ReLU units in a layer separately. First, we calculate pre-activation bounds, i.e., numerical bounds of $z_{i-1,j}, j \in [n_{i-1}], n_{i-1} = n_i$. Let $l_{i-1,j}, u_{i-1,j}$ (l, u for short) denote the lower and upper numerical bounds of $z_{i-1,j}$, i.e., $l_{i-1,j} \leq z_{i-1,j} \leq u_{i-1,j}$. If $l_{i-1,j} \geq 0$ or $u_{i-1,j} \leq 0$, ReLU units degenerate into linear functions ($z_{i,j} = 0$ or $z_{i,j} = z_{i-1,j}$, respectively). In this case, the calculation of parameters in Eq. 5 is similar to that of linear layers. If $l_{i-1,j} \leq 0 \leq u_{i-1,j}$, we need to calculate two bounding lines. The upper bounding line is $z_{i,j} = u(z_{i-1,j} - l)/(u - l)$ while the lower one has various choices.

There are different relaxations for ReLU units, as shown in Fig. 2. The tightest relaxation used in LP-based methods calculate two lower lines, which is time-consuming since each non-linear unit results in an additional bound. Thus, most LiRPA-based methods use the following lower line to minimize the relaxation area (i.e., blue triangles in Fig. 2).

$$\begin{cases} z_{i,j} = 0, \text{if } l + u \geq 0 \text{ (e.g., Fig. 2b)}, \\ z_{i,j} = z_{i-1,j}, \text{otherwise (e.g., Fig. 2c)}. \end{cases} \tag{9}$$

Recently, a few new methods try to tighten ReLU relaxations by optimizing the slopes of lower bound lines (see Fig. 2d), which face challenges in efficiency since their optimization variables (i.e., the slopes) increase with the DNN's scale.

Generally, let $z_{i,j} = k^L_{i,j} z_{i-1,j} + b^L_{i,j}$ and $z_{i,j} = k^U_{i,j} z_{i-1,j} + b^U_{i,j}$ be the lower and upper bounding lines of ReLU unit $z_{i,j}$ respectively, we can get the following parameters in Eq. 5.

$$\begin{aligned} A^{i,L} &= \text{diag}(k^L_{i,1}, k^L_{i,2}, \ldots, k^L_{i,n_i}), \\ A^{i,U} &= \text{diag}(k^U_{i,1}, k^U_{i,2}, \ldots, k^U_{i,n_i}), \\ B^{i,L} &= [b^L_{i,1}, b^L_{i,2}, \ldots, b^L_{i,n_i}], \\ B^{i,U} &= [b^U_{i,1}, b^U_{i,2}, \ldots, b^U_{i,n_i}]. \end{aligned} \tag{10}$$

4 Methodology

Nowadays, the scalability of DNNs proliferates while most works that aim to tighten LiRPA-based methods (e.g., optimizing ReLU bounds or relaxing ReLU units jointly) consume significantly more computational cost, which hinders their scalability. Hence, it is more practical to tighten LiRPA-based

methods without significantly increasing the computational cost. Since the imprecision of LiRPA-based methods is caused by the relaxation of ReLU units, our key insight is to design tighter ReLU relaxations while maintaining adequate efficiency.

As shown in Fig. 2, bounding ReLU units with one lower line always results in a redundant area, while two lower bounding lines can lead to the growth of bounds' number and deterioration of scalability. To tighten the relaxation without significantly hurting the scalability, we propose to calculate an additional pair of lower and upper bounds, which use different relaxations instead of those in the original pair. The benefits of our solution are as follows. First, the numerical bounds calculated from the two pairs can be used for the refinement of each other's pre-activation bounds. Second, the number of linear bounds is limited to ensure adequate efficiency. We introduce the details of our method in the following.

4.1 The Proposed Algorithm: DROWN

Linear Bounds and Propagation. First, we start with building relations between two adjacent layers, e.g., the i-th layer and the $i-1$-th layer. Besides the original pair of bounds, i.e., Eq. 5, we calculate an additional pair of bounds as follows.

$$C^{i,L}_{j,:} z_{i-1} + D^{i,L}_j \leq z_{i,j} \leq C^{i,U}_{j,:} z_{i-1} + D^{i,U}_j, \tag{11}$$

where $C^{i,L/U} \in \mathbb{R}^{n_i \times n_{i-1}}$ and $D^{i,L/U} \in \mathbb{R}^{n_i}$. Similar to the original one (Eq. 7), we can also build relations between $z_{i,j}$ and z_0.

$$C^{0,L}_{j,:} z_0 + D^{0,L}_j \leq z_{i,j} \leq C^{0,U}_{j,:} z_0 + D^{0,U}_j. \tag{12}$$

The two pairs of linear bounds (i.e., Eq. 5 and 11) propagate through the DNNs separately and only interact with each other when performing numerical bounds refinement as shown in the following.

Numerical Bounds Refinement. Since we have that $z_0 \in \mathcal{S}(x_0)$ (e.g., $x_0 - \epsilon \leq z_0 \leq x_0 + \epsilon$), we can substitute it into Eq. 7 to get the numerical bounds of $z_{i,j}$, denoted as $l^1_{i,j}, u^1_{i,j}$ (l^1, u^1 for short), and substitute it into Eq. 12 to get $l^2_{i,j}, u^2_{i,j}$ (l^2, u^2 for short). Note that both l^1, u^1 and l^2, u^2 are calculated by LiRPA, and thus their soundness are guaranteed. Consequently, we have the following equations.

$$\begin{aligned} l'_{i,j} &\leq z_{i,j} \leq u'_{i,j}, \\ l'_{i,j} &= \max\{l^1_{i,j}, l^2_{i,j}\}, \\ u'_{i,j} &= \min\{u^1_{i,j}, u^2_{i,j}\}. \end{aligned} \tag{13}$$

We will refer to $l'_{i,j}, u'_{i,j}$ (l', u' for short) as refined bounds in the rest of this paper.

Linear Bounds of Linear Layers. Assuming that the i-layer is linear and $z_{i,j} = \alpha^i z_{i-1} + \beta^i$, we calculate the parameters in Eq. 11 as follows.

$$C^{i,L} = C^{i,U} = \alpha^i, D^{i,L} = D^{i,U} = \beta^i, \tag{14}$$

which is similar to those in the original pair because there are no relaxations for linear bounds.

Linear Bounds of Non-linear Layers. Here we take ReLU units as an example to demonstrate our calculation procedure, i.e, $z_{i,j} = \text{ReLU}(z_{i-1,j})$. Similar to those in LiRPA, we assume that parameters of Eq. 11 have been calculated for each two adjacent layers before the i-th layer. Here we only discuss the relaxation of ReLU when $l'_{i-1,j} \leq z_{i-1,j} \leq u'_{i-1,j}$ since in other cases the calculation is similar to those for linear layers.

First, we calculate refined bounds l', u' for each $z_{i-i,j}$ in the $(i-1)$-th layer. Note that refined bounds are usually tighter (definitely not looser) than the pre-activation bounds calculated by LiRPA with the original pair of bounds, which is beneficial to the tightening of relaxations, as shown in Fig. 1.

Then, we calculate bounding lines for the two pairs of linear bounds according to the refined pre-activation bounds l', u'. The upper bounding lines of the two pairs is identical, denoted as $z_{i,j} = k^{U,1}_{i,j} z_{i-1,j} + b^{U,1}_{i,j}$.

$$k^{U,1}_{i,j} = \frac{u'}{u' - l'}, \; b^{U,1}_{i,j} = \frac{u'l'}{u' - l'}. \tag{15}$$

For the lower bounding lines, we set different bounding lines for each pair of bounds. Let $z_{i,j} = k^{L,1}_{i,j} z_{i-1,j} + b^{L,1}_{i,j}$ and $z_{i,j} = k^{L,2}_{i,j} z_{i-1,j} + b^{L,2}_{i,j}$ be the two lower bounding lines respectively.

$$\begin{aligned} k^{L,1}_{i,j} &= 0, b^{L,1}_{i,j} = 0, \text{ (e.g., Fig. 2b)} \\ k^{L,2}_{i,j} &= 1, b^{L,2}_{i,j} = 0. \text{ (e.g., Fig. 2c)} \end{aligned} \tag{16}$$

Finally, the parameters of the two pairs of bounding lines (Eq. 5 and Eq. 11) are calculated according to the following equations.

$$\begin{aligned} A^{i,L} &= \text{diag}(k^{L,1}_{i,1}, k^{L,1}_{i,2}, \ldots, k^{L,1}_{i,n_i}), \\ C^{i,L} &= \text{diag}(k^{L,2}_{i,1}, k^{L,2}_{i,2}, \ldots, k^{L,2}_{i,n_i}), \\ A^{i,U} &= C^{i,U} = \text{diag}(k^{U,1}_{i,1}, k^{U,1}_{i,2}, \ldots, k^{U,1}_{i,n_i}), \\ B^{i,L} &= [b^{L,1}_{i,1}, b^{L,1}_{i,2}, \ldots, b^{L,1}_{i,n_i}], \\ D^{i,L} &= [b^{L,2}_{i,1}, b^{L,2}_{i,2}, \ldots, b^{L,2}_{i,n_i}], \\ B^{i,U} &= D^{i,U} = [b^{U,1}_{i,1}, b^{U,1}_{i,2}, \ldots, b^{U,1}_{i,n_i}]. \end{aligned} \tag{17}$$

Algorithm 1: DROWN

Input: a model f with N layers, a sample x_0, its label y_t, and an adversarial space $\mathcal{S}(x_0)$
Output: certification result

Initial a list $\mathcal{L}$ to store linear bound parameters;
for $i \leftarrow 1$ **to** N **do**
  if *the i-th layer is linear* **then**
    Calculate linear bound parameters according to Eq. 8 and 14;
    Append the i-th layer's linear bound parameters to $\mathcal{L}$;
  else
    Calculate numerical bounds $l^1_{i-1,j}, u^1_{i-1,j}$ using $\mathcal{S}(x_0)$ and $A^{i,L/U}, B^{i,L/U}$ in $\mathcal{L}$;
    Calculate numerical bounds $l^2_{i-1,j}, u^2_{i-1,j}$ using $\mathcal{S}(x_0)$ and $C^{i,L/U}, D^{i,L/U}$ in $\mathcal{L}$;
    Calculate refined bounds $l'_{i-1,j}, u'_{i-1,j}$ according to Eq. 13;
    Calculate bounding lines according to Eq. 15 and 16;
    Calculate linear bound parameters according to Eq. 17;
    Append the i-th layer's linear bound parameters to $\mathcal{L}$;
Calculate numerical bounds $l^1_{N,j}, u^1_{N,j}$ using the first pair of linear bounds in $\mathcal{L}$;
Calculate numerical bounds $l^2_{N,j}, u^2_{N,j}$ using the second pair of linear bounds in $\mathcal{L}$;
Calculate refined bounds $l'_{N,j}, u'_{N,j}$ according to Eq. 13;
$l \leftarrow l'_N, u \leftarrow u'_N$;
if l, u *satisfy the condition in Eq. 4* **then**
  return `Certified`;
else
  return `Uncertain`;

The complete algorithm is shown in Algorithm 1. We refer to $A^{i,L/U}, B^{i,L/U}, C^{i,L/U}, D^{i,L/U}$ as linear bound parameters. The first pair of bounds refer to linear bounds with $A^{i,L/U}, B^{i,L/U}$ and the second pair refer to those with $C^{i,L/U}, D^{i,L/U}$. We provide a simple example of DROWN with its comparison to CROWN in Appendix A.

4.2 Discussion

Effectiveness. DROWN's effectiveness primarily stems from its refined pre-activation bounds. As Fig. 1 illustrates, the relaxed area of the refined bound, i.e., the green triangle, is smaller than that of the original bounds, i.e., the blue triangle. Smaller relaxed areas are crucial to the tightening of robustness certification because the linear bounds after the relaxation will be closer to the exact ones. We also demonstrate DROWN's effectiveness in our experimental evaluations.

Efficiency. According to Zhang et al. (2018), CROWN's time complexity is $O(m^2n^3)$ for an m-layer model with n neurons in each layer. Since DROWN doubles CROWN's calculations, its time complexity is also $O(m^2n^3)$. Though it's challenging to certify large models, we note that even very small models have practical applications in specific domains such as IoT devices (Rjoub et al., 2024).

Beyond ReLU Relaxation. This paper focuses on the relaxation of activations. Thus, we extend DROWN to other widely-adopted activations such as LReLU (similarly, PReLU and RReLU), ELU (similarly, CELU and SELU), and Swish (similarly, Mish and Hardswish). Similar to the two different relaxations for ReLU (e.g., Fig. 2b and 2c), we present illustrations of those for LReLU, ELU, and Swish in Fig. 3. Due to limited space, the details of those relaxations are presented in Appendix C.

More discussions are provided in Appendix B.

5 Experiments

5.1 Setup

Datasets. We use the following datasets for evaluation. *MNIST* dataset (LeCun et al., 1998), which is a famous image classification dataset. *SST* dataset (Socher et al., 2013), which is a benchmark corpus of movie reviews for sentiment analysis. *Yelp* dataset (Zhang et al., 2015), which is a large text classification benchmark.

Models. We use the following models for evaluation. *Convolutional Neural Network* models, which

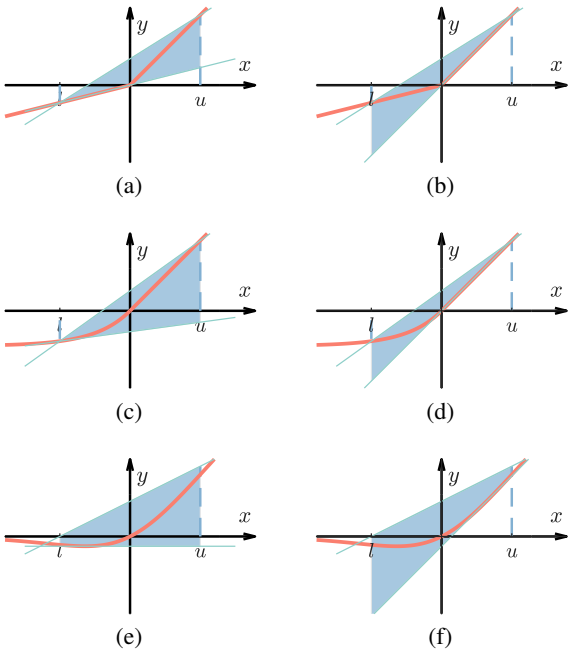


Figure 3: The relaxations for LReLU (a,b), ELU (c,d), and Swish (e,f) used in DROWN.

Method	CROWN		DROWN	
Model	C.Acc.	Time	C.Acc.	Time
CNN1	46.83%	0.008	**84.84%**	0.012
CNN2	48.35%	0.008	**88.96%**	0.013
CNN3	10.42%	0.037	**87.96%**	0.072
CNN4	6.92%	0.074	**90.31%**	0.147
CNN5	13.51%	0.044	**86.96%**	0.085
CNN6	8.95%	0.084	**90.74%**	0.167
CNN7	35.90%	0.038	**86.12%**	0.073
CNN8	37.35%	0.117	**87.15%**	0.232
CNN9	61.73%	0.033	**82.07%**	0.063
CNN10	35.87%	0.086	**89.49%**	0.169

Table 1: Evaluation results of CROWN and DROWN on 10 CNNs trained on the MNIST dataset ($\epsilon = 0.3$ under the ℓ_∞ norm). All time results are in seconds.

are classic models for image classification. We use CNN models with 4 to 7 layers. *Transformer* models, which are the most widely applied models for NLP tasks. We use a Transformer with 3 layers, 4 attention heads, 128 hidden neurons (the feed-forward network in it also has 128 hidden neurons). We also conduct evaluations on RNNs (see Appendix D), which are presented in Appendix F.

Evaluation Metrics. We use the following metrics for evaluation: The *certified robust accuracy* (C.Acc.), which is the fraction of test samples for which the robustness certification method confirms that the model's predictions within the given region around them are robust. The *certified radius* (also referred to as certified bounds), which is the radius of the largest ℓ_p-ball within which the model's prediction is proved to be robust by a robustness certification method.

We provide additional details in Appendix E.

5.2 CNN Results

In this subsection, we evaluate the precision and efficiency on CNNs. We certify the robustness of 10 CNNs with various depths and widths against ℓ_p-bounded attackers, i.e., attackers who try to search adversarial examples within a neighborhood $\{x : \|x - x_0\|_p \leq \epsilon\}$ around the clean sample x_0. We report certified robust accuracy and average running time per batch in Tab. 1.

Precision. The certified robust accuracy calculated by DROWN is higher on all models. As shown in Tab. 1, DROWN achieves 20.34% to 83.39% higher certified robust accuracy than CROWN. The results above indicate that DROWN is more precise than the baseline method.

Efficiency. Though DROWN is slower, its computational cost is acceptable. As shown in Tab. 1, the running time of DROWN is about twice that of CROWN, which is reasonable since our method propagates two pairs of linear bounds while CROWN propagates one. The results above indicate that the scalability of DROWN is similar to CROWN's.

5.3 Transformer Results

In this subsection, we extend DROWN to support Transformers and evaluate its precision and efficiency under two scenarios. In the one-word attacks scenario, we assume that attackers can perturb arbitrary but only one word's embedding (within a certain ℓ_p-bounded neighborhood around the embedding vector) in a sentence. In the multi-word attacks scenario, we suppose more powerful attackers can perturb multiple word's embedding simultaneously (also ℓ_p-bounded). Due to limited GPU memory, we limit multi-word attackers to perturbing at most four words in the sentence. We report the minimum of certified radii, the average of certified radii, the average of running time, and the average of GPU memory usage in Tab. 2 and 3.

Precision. Both the minimum and the average of certified radii calculated by DROWN are the

Dataset		SST				Yelp			
Method	Norm	Min.	Avg.	Time(s)	Mem.(B)	Min.	Avg.	Time	Mem.
CROWN-BaF	ℓ_1	0.036	1.686	3.7	5.4e8	0.102	0.573	3.6	4.8e8
	ℓ_2	6.1e-3	0.328	3.7	5.4e8	0.023	0.137	3.5	4.9e8
	ℓ_∞	6.0e-4	0.033	3.7	5.4e8	2.4e-3	0.015	3.8	5.1e8
DeepT	ℓ_1	0.036	1.806	24.8	3.9e9	0.077	0.427	26.2	3.0e9
	ℓ_2	6.4e-3	0.330	25.4	3.9e9	0.068	0.186	26.1	3.0e9
	ℓ_∞	2.0e-3	0.032	24.7	3.9e9	7.9e-3	0.024	25.1	3.0e9
GaLileo	ℓ_1	0.036	1.749	3.7	5.4e8	0.262	0.755	3.4	4.8e8
	ℓ_2	6.2e-3	0.337	3.7	5.4e8	0.059	0.181	3.5	4.8e8
	ℓ_∞	6.0e-4	0.034	3.7	5.4e8	6.2e-3	0.020	3.6	4.8e8
DROWN	ℓ_1	0.178	**3.064**	5.6	4.7e9	0.836	**2.638**	5.7	3.7e9
	ℓ_2	0.034	**0.569**	5.9	4.7e9	0.163	**0.638**	5.7	3.8e9
	ℓ_∞	3.3e-3	**0.056**	5.5	4.6e9	0.017	**0.070**	5.7	3.7e9

Table 2: Evaluation results of CROWN-BaF (Shi et al., 2020), DeepT (Bonaert et al., 2021), GaLileo (Zhang et al., 2024), and DROWN on Transformers trained on the SST and Yelp datasets against one-word attacks.

highest among all methods. As shown in Tab. 2, DROWN achieves 1.65 to 3.49 times larger certified radii than baseline methods against one-word attacks. Besides, as we can seen in Tab. 3, DROWN achieves 2.00 to 4.68 times larger certified radii than baseline methods against multi-word attacks. The results above show that DROWN is more precise than baseline methods.

Efficiency. DROWN is slower than the fastest method but its computational cost is acceptable. As shown in Tab. 2 and 3, the average of DROWN's running time is about two times longer than that of CROWN-BaF or GaLileo but DROWN is still significantly faster than DeepT. As for GPU memory usage, DROWN is the most memory-consuming method. As shown in Tab. 2 and 3, DROWN requires up to 4.17 times more memory than DeepT, the second most memory-consuming method. Considering that DROWN is more precise than baseline methods, we conclude that DROWN is the best choice for robustness certification of Transformers.

5.4 Beyond Norm-bounded Attacks

In addition to evaluations against ℓ_p-bounded attacks above, we further conduct evaluation against synonym substitution attacks where every word in the sentence can be replaced with one of its synonyms, which is considered to be more intuitive. Following previous works (Bonaert et al., 2021), we record the number of certified sentences of DROWN and compare it to previous methods that are also based on LiRPA, which are shown in Tab. 4. The model used here is a 3-layer Transformer with 4 attention heads and an embedding size of 64. The result demonstrates that DROWN is more precise than previous methods at the cost of approximately double the running time.

5.5 Beyond ReLU Activations

To further evaluate DROWN's performance on other activations, we record its certified accuracy and running time on seven CNNs with three widely-applied activation functions (i.e., LReLU, ELU, and Swish) respectively and compare its results to CROWN's. As shown in Tab. 5, DROWN achieves higher precision than CROWN, albeit with approximately double the running time.

6 Conclusion

In this paper, we present DROWN, a novel method to tighten bounds in LiRPA-based robustness certification without significantly reducing the scalability. DROWN propagates two pairs of linear bounds using two different relaxations to efficiently refine pre-activation bounds of ReLU units in later layers. Our evaluation results across various models and datasets show that DROWN achieves up to 83.39% higher certified robust accuracy than the baseline on CNNs and up to 4.68 times larger certified radii than the baseline on Transformers while consuming about twice the baseline's running time.

Dataset		SST				Yelp			
Method	Norm	Min.	Avg.	Time(s)	Mem.(B)	Min.	Avg.	Time	Mem.
CROWN-BaF	ℓ_1	6.7e-3	0.391	5.5	1.0e9	1.1e-3	0.101	4.5	9.6e8
	ℓ_2	1.2e-3	0.076	4.7	1.0e9	2.4e-4	0.025	4.3	9.3e8
	ℓ_∞	1.1e-4	7.6e-3	5.2	1.0e9	2.6e-5	2.7e-3	5.1	1.6e9
DeepT	ℓ_1	6.2e-4	9.7e-3	27.0	3.0e9	4.6e-5	7.4e-3	28.1	2.3e9
	ℓ_2	6.2e-4	9.5e-4	27.3	3.0e9	3.4e-5	7.2e-3	28.1	2.3e9
	ℓ_∞	5.8e-4	7.4e-3	27.6	3.0e9	1.3e-5	5.6e-3	28.2	2.2e9
GaLileo	ℓ_1	6.7e-3	0.410	5.4	1.0e9	1.2e-3	0.170	4.5	9.5e8
	ℓ_2	1.2e-3	0.080	4.6	1.0e9	2.7e-4	0.043	4.2	9.2e8
	ℓ_∞	1.1e-4	8.0e-3	5.2	1.0e9	2.9e-5	4.9e-3	5.1	9.8e8
DROWN	ℓ_1	0.040	**0.895**	9.6	1.0e10	0.204	**0.796**	9.1	9.6e9
	ℓ_2	7.5e-3	**0.160**	8.5	1.0e10	0.058	**0.195**	7.6	8.9e9
	ℓ_∞	7.4e-4	**0.016**	8.1	9.7e9	4.6e-3	**0.021**	7.3	8.8e9

Table 3: Evaluation results of CROWN-BaF, DeepT, GaLileo, and DROWN on Transformers trained on the SST and Yelp datasets against multi-word attacks.

Method	Certified Sentences	Time(s)
CROWN-BaF	133	0.80
GaLileo	133	0.75
DROWN	134	1.47

Table 4: Evaluation results of CROWN-BaF, GaLileo, and DROWN against synonym substitution attacks on 135 sentences in the SST dataset.

Method	CROWN		DROWN	
Model	C.Acc.	Time	C.Acc.	Time
LReLU-1	51.38%	2.205	**86.17%**	4.283
LReLU-2	26.19%	1.519	**88.17%**	3.051
LReLU-3	49.77%	2.704	**85.31%**	5.203
ELU-1	51.44%	2.251	**84.04%**	4.564
ELU-2	89.91%	1.675	**94.58%**	3.154
ELU-3	80.42%	2.675	**89.17%**	5.397
Swish-1	20.85%	2.262	**46.32%**	4.456

Table 5: Evaluation results of CROWN and DROWN on 7 CNNs (with LReLU, ELU, and Swish as activation respectively) trained on the MNIST dataset. All time results are in seconds.

7 Limitations

This study has certain important potential limitations. First, similar to other LiRPA-based methods, DROWN is challenged by modern pre-trained models. In fact, most deterministic methods for robustness certification are challenged by the scale of pre-trained models. Currently, only probabilistic methods (or randomized smoothing) can certify such large models, which build a smoothed model with higher robustness upon a given model. However, the smoothed model involves a large number of random sampling, which leads to an increase in computational cost in the inference phase (deterministic methods do not affect the inference phase). Second, due to relaxations, certifying nontrivial robustness for very deep models requires further research. Specifically, the errors caused by relaxations accumulate layer by layer, which enlarges the pre-activation bounds and eventually results in trivial results for deeper models. DROWN can only mitigate the above issue. Finally, like most of previous works, DROWN is restricted to classification task. The reason is that the current definition of robustness (i.e., Eq. 1) is tailored for classification tasks and there is no well-established definitions of robustness for other tasks (albeit a few recent works that propose their own definitions of robustness for tasks beyond classification such as semantic segmentation (Yatsura et al., 2023)).

Acknowledgments

This work was supported by the National Natural Science Foundation of China (under Grant No. 62372268), the Major Scientific and Technological Innovation Projects of Shandong Province (under Grant No. 2024CXGC010114), Shandong Provincial Natural Science Foundation (under Grant No. ZR2022LZH013 and No. ZR2021LZH007), and Jinan City "20 New Universities" Funding Project (under Grant No. 2021GXRC084) and partly supported by the National Natural Science Foundation of China (under Grant No. 62402418) and the Key R&D Program of Ningbo (under Grant No. 2024Z115).

References

Gregory Bonaert, Dimitar I. Dimitrov, Maximilian Baader, and Martin T. Vechev. 2021. Fast and precise certification of transformers. In *PLDI*, pages 466–481.

Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. 2019. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, pages 3240–3247.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. 2018. A unified view of piecewise linear neural network verification. In *NeurIPS*, pages 4795–4804.

Nicholas Carlini and David A. Wagner. 2017. Towards evaluating the robustness of neural networks. In *S&P*, pages 39–57.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for natural language inference. In *ACL*, pages 1657–1668.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. 2017. Maximum resilience of artificial neural networks. In *ATVA*, volume 10482, pages 251–268.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, pages 103–111.

Tianyu Du, Shouling Ji, Lujia Shen, Yao Zhang, Jinfeng Li, Jie Shi, Chengfang Fang, Jianwei Yin, Raheem Beyah, and Ting Wang. 2021. Cert-rnn: Towards certifying the robustness of recurrent neural networks. In *CCS*, pages 516–534.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. 2018. A dual approach to scalable verification of deep networks. In *UAI*, pages 550–559.

Rüdiger Ehlers. 2017. Formal verification of piecewise linear feed-forward neural networks. In *ATVA*, volume 10482, pages 269–286.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: safety and robustness certification of neural networks with abstract interpretation. In *S&P*, pages 3–18.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *ICLR*.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. 2019. Scalable verified training for provably robust image classification. In *ICCV*, pages 4841–4850.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9:1735–1780.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, volume 10426, pages 97–117.

Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019. POPQORN: quantifying robustness of recurrent neural networks. In *ICML*, volume 97, pages 3468–3477.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *EMNLP*, pages 4470–4481.

Linyi Li, Xiangyu Qi, Tao Xie, and Bo Li. 2020. Sok: Certified robustness for deep neural networks. *CoRR*, abs/2009.04131.

Linyi Li, Tao Xie, and Bo Li. 2023. Sok: Certified robustness for deep neural networks. In *S&P*, pages 1289–1310.

Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351.

Jingyue Lu and M. Pawan Kumar. 2020. Neural network branching for neural network verification. In *ICLR*.

Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. 2020. Fastened CROWN: tightened neural network robustness certificates. In *AAAI*, pages 5037–5044.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.

Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, volume 80, pages 3575–3583.

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.*, 6(POPL):1–33.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, pages 115–124.

Gaith Rjoub, Saidul Islam, Jamal Bentahar, Mohammed Amin Almaiah, and Rana Al-Rawashdeh. 2024. Enhancing iot intelligence: A transformer-based reinforcement learning methodology. *CoRR*, abs/2404.04205.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A convex relaxation barrier to tight robustness verification of neural networks. In *NeurIPS*, pages 9832–9842.

Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. 2020. Robustness verification for transformers. In *ICLR*.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. 2019a. Beyond the single neuron convex barrier for neural network certification. In *NeurIPS*, pages 15072–15083.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and effective robustness certification. In *NeurIPS*, pages 10825–10836.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019b. An abstract domain for certifying neural networks. *PACMPL*, 3:41:1–41:30.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. 2020. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. In *NeurIPS*.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*, pages 5998–6008.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards fast computation of certified robustness for relu networks. In *ICML*, volume 80, pages 5273–5282.

Eric Wong and J. Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, volume 80, pages 5283–5292.

Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. 2018. Scaling provable adversarial defenses. In *NeurIPS*, pages 8410–8419.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. In *NeurIPS*.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *ICLR*.

Maksym Yatsura, Kaspar Sakmann, N. Grace Hua, Matthias Hein, and Jan Hendrik Metzen. 2023. Certified defences against adversarial patch attacks on semantic segmentation. In *ICLR*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *NeurIPS*, pages 4944–4953.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NeurIPS*, pages 649–657.

Yunruo Zhang, Lujia Shen, Shanqing Guo, and Shouling Ji. 2024. Galileo: General linear relaxation framework for tightening robustness certification of transformers. In *AAAI*, pages 21797–21805.

Zhaodi Zhang, Yiting Wu, Si Liu, Jing Liu, and Min Zhang. 2022. Provably tightest linear approximation for robustness verification of sigmoid-like neural networks. In *ASE*, pages 80:1–80:13.

A A Simple Example of DROWN

We provide a simple example of DROWN to facilitate understanding. As shown in Fig. 4, $x_1, \ldots, x_6$ are six neurons in a hypothetical DNN. The forward process of that DNN is

$$x_3 = x_1 + x_2, \tag{18}$$
$$x_4 = x_1 - x_2, \tag{19}$$
$$x_5 = \text{ReLU}(x_3), \tag{20}$$
$$x_6 = \text{ReLU}(x_4), \tag{21}$$

where $-\frac{1}{2} \leq x_1 \leq 1$ and $-\frac{1}{2} \leq x_2 \leq 1$. We show the bounding procedure and results of CROWN and DROWN in the following.

CROWN Bounds. We start with explaining how CROWN derive the numerical bounds of x_5 and x_6. The first layer of the DNN (i.e., x_3 and x_4) is a linear layer, which does not need relaxations. Thus, according to Eq. 18 and 19, the linear bounds of x_3 and x_4 are

$$x_1 + x_2 \leq x_3 \leq x_1 + x_2, \tag{22}$$
$$x_1 - x_2 \leq x_4 \leq x_1 - x_2. \tag{23}$$

The second layer of the DNN (i.e., x_5 and x_6) is a non-linear layer, which requires relaxations. To perform relaxations for x_5 and x_6, we need to calculate their pre-activation bounds first, i.e., numerical bounds of x_3 and x_4. Given the numerical bounds of x_1 and x_2, we can derive the following numerical bounds based on Eq. 22 and 23.

$$-1 \leq x_3 \leq 2,\ -\frac{3}{2} \leq x_4 \leq \frac{3}{2}. \tag{24}$$

According to the ReLU relaxation in CROWN (i.e., Eq. 9), the upper and lower bound of x_5 and x_6 are

$$x_3 \leq x_5 \leq \frac{2}{3}x_3 + \frac{2}{3}, \tag{25}$$
$$0 \leq x_6 \leq \frac{1}{2}x_4 + \frac{3}{4}. \tag{26}$$

We substitute x_3 and x_4 in Eq. 25 and 26 with Eq. 22 and 23 (i.e., the backward propagation of the bounds) to derive the following linear bounds.

$$x_1 + x_2 \leq x_5 \leq \frac{2}{3}(x_1 + x_2) + \frac{2}{3}, \tag{27}$$
$$0 \leq x_6 \leq \frac{1}{2}(x_1 - x_2) + \frac{3}{4}. \tag{28}$$

Based on the numerical bounds of x_1 and x_2, we can derive the numerical bounds of x_5 and x_6 as follows.

$$-1 \leq x_5 \leq 2,\ 0 \leq x_6 \leq \frac{3}{2}. \tag{29}$$

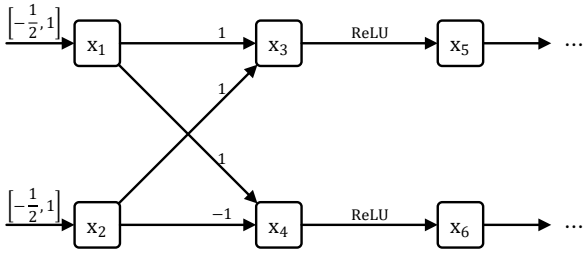


Figure 4: A hypothetical DNN as the example to explain DROWN (as well as CROWN) in detail.

DROWN Bounds. In contrast to CROWN, DROWN propagates two pairs of linear bounds. For linear layer such as the first layer in the hypothetical DNN, the two pairs of linear bounds are identical (i.e., Eq. 22 and 23). Since they are identical to those in CROWN, we do not rewrite the same equations here.

For non-linear layer such as the second layer, DROWN calculates two pairs of linear bounds that are different from each other. Since the second layer is the first non-linear layer in the DNN, the pre-activation bounds of x_5 and x_6 are identical to those calculated by CROWN (i.e., Eq. 24). Note that the pre-activation bounds will be different from CROWN's after the first non-linear layer and different from each other. Thus, DROWN refines the pre-activation bounds using Eq. 13.

Based on the pre-activation bounds, two pairs of linear bounds are calculated. The first pair of linear bounds for x_5 and x_6 are

$$0 \leq x_5 \leq \frac{2}{3}x_3 + \frac{2}{3}, \tag{30}$$
$$0 \leq x_6 \leq \frac{1}{2}x_4 + \frac{3}{4}. \tag{31}$$

The second pair of linear bounds for x_5 and x_6 are

$$x_3 \leq x_5 \leq \frac{2}{3}x_3 + \frac{2}{3}, \tag{32}$$
$$x_4 \leq x_6 \leq \frac{1}{2}x_4 + \frac{3}{4}. \tag{33}$$

By substituting x_3 and x_4 in the two pairs of linear bounds with the two pairs of linear bounds in the last layer respectively (i.e., the backward propagation of linear bounds), we derive the following linear bounds. The first pair of linear bounds for x_5 and x_6 (with respect to x_1 and x_2) are

$$0 \leq x_5 \leq \frac{2}{3}(x_1 + x_2) + \frac{2}{3}, \tag{34}$$
$$0 \leq x_6 \leq \frac{1}{2}(x_1 - x_2) + \frac{3}{4}. \tag{35}$$

The second pair of linear bounds for x_5 and x_6 (with respect to x_1 and x_2) are

$$x_1 + x_2 \leq x_5 \leq \frac{2}{3}(x_1 + x_2) + \frac{2}{3}, \tag{36}$$

$$x_1 - x_2 \leq x_6 \leq \frac{1}{2}(x_1 - x_2) + \frac{3}{4}. \tag{37}$$

Based on the numerical bounds of x_1 and x_2, we can derive the two pairs of numerical bounds of x_5 and x_6 as follows.

$$0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq \frac{3}{2}, \tag{38}$$

$$-1 \leq x_5 \leq 2, \quad -\frac{3}{2} \leq x_6 \leq \frac{3}{2}. \tag{39}$$

DROWN refines the numerical bounds of x_5 and x_6 with Eq. 13 and derives the following refined numerical bounds.

$$0 \leq x_5 \leq 2,\ 0 \leq x_6 \leq \frac{3}{2}. \tag{40}$$

Comparison. Compared to CROWN's results (i.e., Eq. 29), DROWN's results for x_5 is tighter while its results for x_6 is identical to CROWN's, which indicates that DROWN can tighten some (not all) relaxations in the first layer of DNNs. The results of tighter relaxations (e.g., the refined numerical bounds of x_5 and x_6) will result in tighter pre-activation bounds of the later non-linear layers, which can further tighten the relaxations in those layers (as shown in Fig. 1).

B Additional Discussions

Why not use more pairs of linear bounds? DROWN only use two pairs of linear bounds because they're sufficient in ReLU relaxations. For simplicity, we only analyze the relationship between the slope of the lower bounding line $k^L_{i,j}$ and the corresponding units $z_{i,j}$ here. According to the previous equations, we have that

$$C^{i,L}_{j,j} z_{i-1,j} + D^{i,L}_j \leq z_{i,j}, \tag{41}$$

where $C^{i,L}_{j,j} = k^L_{i,j}$ and $D^{i,L}_j = 0$. As shown in Fig. 2, the range of $k^L_{i,j}$ is $0 \leq k^L_{i,j} \leq 1$. Meanwhile, we don't know whether $z_{i-1,j}$ is positive or negative. Apparently, $z_{i,j}$ reaches it minimum when $k^L_{i,j} = 0$ or 1. Therefore, we set $k^L_{i,j} = 0$ for one pair of linear bounds and $k^L_{i,j} = 1$ for the other, as shown in Eq. 16. This conclusion also holds for LReLU, PReLU, and RReLU.

Beyond LiRPA-based methods. The methodology of DROWN can be used to tighten some other robustness certification methods. For example, we extend DROWN to zonotope-based methods and evaluate it on some RNNs, which is presented in Appendix F.

C Extension to More Activations

We extend DROWN to other widely-adopted activations such as LReLU (which shares similar properties with PReLU and RReLU), ELU (which shares similar properties with CELU and SELU), and Swish (which shares similar properties with Mish and Hardswish). Similar to ReLU, we provide the upper and lower bounds of those activations (denoted by $y = \sigma(x)$). For an activation neuron $z_{i,j} = \sigma(z_{i-1,j})$ where $l'_{i-1,j} \leq z_{i-1,j} \leq u'_{i-1,j}$ (l', u' for short), we calculate two pairs of linear bounds as follows (note that l', u' here are the refined bounds).

$$\begin{cases} z_{i,j} = k^{L,1}_{i,j} z_{i-1,j} + b^{L,1}_{i,j}, \\ z_{i,j} = k^{U,1}_{i,j} z_{i-1,j} + b^{U,1}_{i,j}, \end{cases} \quad \text{first pair} \tag{42}$$

$$\begin{cases} z_{i,j} = k^{L,2}_{i,j} z_{i-1,j} + b^{L,2}_{i,j}, \\ z_{i,j} = k^{U,2}_{i,j} z_{i-1,j} + b^{U,2}_{i,j}, \end{cases} \quad \text{second pair} \tag{43}$$

where $k^{U,2}_{i,j} = k^{U,1}_{i,j}$ and $b^{U,2}_{i,j} = b^{U,1}_{i,j}$ if using a shared upper bound as used for relaxing ReLU. The two pairs of linear bounds should satisfy the following condition.

$$\begin{aligned} k^{L,1/2}_{i,j} z_{i-1,j} + b^{L,1/2}_{i,j} &\leq \sigma(z_{i-1,j}) \\ &\leq k^{U,1/2}_{i,j} z_{i-1,j} + b^{U,1/2}_{i,j}, \\ &\forall z_{i-1,j} \in [l', u'], \end{aligned} \tag{44}$$

where $1/2$ means the condition holds for the first and second pairs of bounds.

We provide the detailed calculation of $k^{L/U,1/2}_{i,j}$ and $b^{L/U,1/2}_{i,j}$ for each activation in the following.

LReLU. The LReLU activation function is

$$\text{LReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha x, & \text{otherwise}, \end{cases} \tag{45}$$

where α is the negative slope (default: $\alpha = 0.01$). Similar to ReLU, LReLU (and PReLU, RReLU) is a piece-wise linear function. Thus, we only need to relax it when $l' \leq 0 \leq u'$.

We use a shared upper bound for LReLU as follows.

$$k_{i,j}^{U,1} = \frac{\text{LReLU}(u') - \text{LReLU}(l')}{u' - l'}, \quad b_{i,j}^{U,1} = u'(1 - k_{i,j}^{U,1}). \tag{46}$$

The lower bounds in the first and second pairs of linear bounds for LReLU are

$$k_{i,j}^{L,1} = \alpha, \ \ b_{i,j}^{L,1} = 0, \text{ (e.g., Fig. 3a)} \quad k_{i,j}^{L,2} = 1, \ \ b_{i,j}^{L,2} = 0. \text{ (e.g., Fig. 3b)} \tag{47}$$

ELU. The ELU activation function is

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha(e^x - 1), & \text{otherwise}, \end{cases} \tag{48}$$

where α is a hyper-parameter (default: $\alpha = 1.0$). The right half of ELU is linear while its left half is not. Thus, we only need to relax it when $l' \leq 0$.

We use a shared upper bound for LReLU as follows.

$$k_{i,j}^{U,1} = \frac{\text{ELU}(u') - \text{ELU}(l')}{u' - l'}, \quad b_{i,j}^{U,1} = \text{ELU}(u') - u' \cdot k_{i,j}^{U,1}. \tag{49}$$

The lower bounds in the first and second pairs of linear bounds for LReLU are

$$\begin{aligned} k_{i,j}^{L,1} &= \text{ELU}'(l'), \text{(e.g., Fig. 3c)} \\ b_{i,j}^{L,1} &= \text{ELU}(l') - l' \cdot \text{ELU}'(l'), \\ k_{i,j}^{L,2} &= \text{ELU}'(u'), \text{(e.g., Fig. 3d)} \\ b_{i,j}^{L,2} &= \text{ELU}(u') - u' \cdot \text{ELU}'(u'), \end{aligned} \tag{50}$$

where ELU' is the derivative of ELU.

Swish. The Swish (also known as SiLU) activation function is

$$\text{Swish}(x) = x \cdot \text{Sigmoid}(x). \tag{51}$$

The direct relaxation of Swish is more complicated since it is non-convex (e.g., the straight line between $(l, \text{Swish}(l))$ and $(u, \text{Swish}(u))$ may not be its upper bound). However, we discover a globally upper and lower bounds of Swish (as the following equation shows), which are piece-wise linear, as shown in Fig. 5.

$$\text{ReLU}(x) + \beta \leq \text{Swish}(x) \leq \text{ReLU}(x). \tag{52}$$

Calculating the exact value of β is challenging (and unnecessary) since it involves solving transcendental equations. Thus, we calculate a lower bound of β, i.e., $\underline{\beta} = -0.2786$.

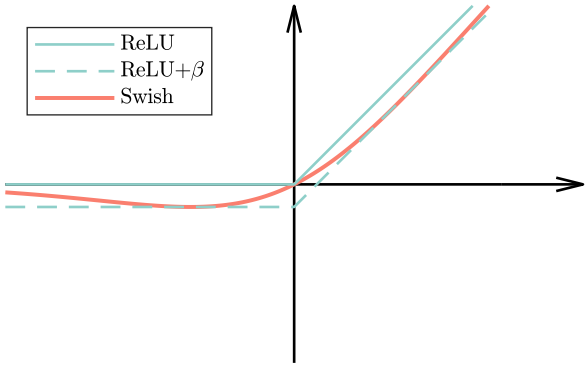


Figure 5: Piece-wise linear bounds in global relaxation of Swish (where we set $\beta = -0.2786$).

We will discuss the relaxed relaxation of Swish in three different cases.

Case 1: $0 \leq l' \leq u'$. In this case, we use a shared upper bound as well as a shared lower bound as follows (i.e., the upper bounds for the two pairs of bounds are identical, so do the lower bounds).

$$k_{i,j}^{U,1} = 1, \ b_{i,j}^{U,1} = 0, \tag{53}$$

$$k_{i,j}^{L,1} = 1, \ b_{i,j}^{L,1} = \beta. \tag{54}$$

Case 2: $l' \leq u' \leq 0$. In this case, we also use a shared upper bound as well as a shared lower bound (in other words, we set $k_{i,j}^{U,1} = k_{i,j}^{U,2}, b_{i,j}^{U,1} = b_{i,j}^{U,2}, k_{i,j}^{L,1} = k_{i,j}^{L,2}, b_{i,j}^{L,1} = b_{i,j}^{L,2}$).

$$k_{i,j}^{U,1} = 0, \ b_{i,j}^{U,1} = 0, \tag{55}$$

$$k_{i,j}^{L,1} = 0, \ b_{i,j}^{L,1} = \beta. \tag{56}$$

Case 3: $l' \leq 0 \leq u'$. In this case, we adopt the upper bound of ReLU (i.e., Swish's globally upper bound) as the shared upper bound, which is presented in the following.

$$k_{i,j}^{U,1} = \frac{u'}{u' - l'}, \ b_{i,j}^{U,1} = \frac{u'l'}{u' - l'}. \tag{57}$$

Meanwhile, we use the lower bounds of Swish's globally lower bound (i.e., $\text{ReLU}(x) + \beta$) as the lower bounds of Swish respectively as follows.

$$k_{i,j}^{L,1} = 0, b_{i,j}^{L,1} = \beta, \text{ (e.g., Fig. 3e)} \quad k_{i,j}^{L,2} = 1, b_{i,j}^{L,2} = \beta. \text{ (e.g., Fig. 3f)} \tag{58}$$

D RNNs for Text Classification

Following prior works (Ko et al., 2019; Du et al., 2021), we apply RNNs to text classification tasks

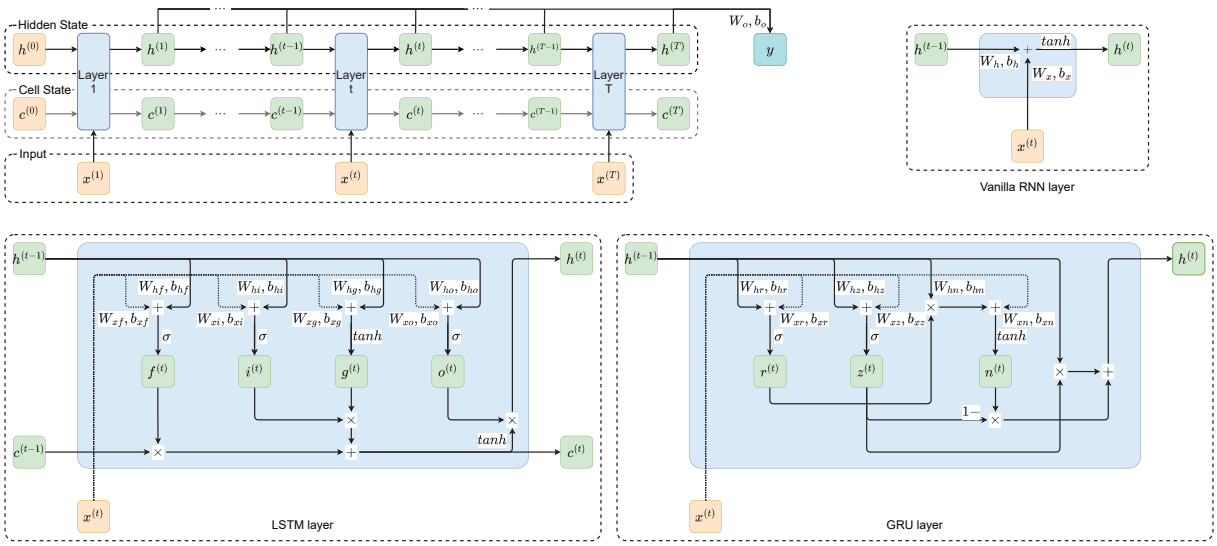


Figure 6: The recurrent neural network architecture for text classification. The cell state is unique to LSTM models.

such as sentiment prediction and toxic content detection. The RNN model receives a sentence of variable lengths (denoted by T) composed of words (or tokens) and performs classification into C distinct classes. As shown in Fig. 6, the RNN model updates a hidden state h_t at each time step according to the current frame x_t and the last hidden state h_{t-1}:

$$h_t = \text{layer}(x_t, h_{t-1}), \tag{59}$$

where $t = 1, 2, \ldots, T$. The initial hidden state h_0 is usually set to zero. In particular, LSTM models update an additional hidden state c_t called cell state. The output vector $h \in \mathbb{R}^C$ is calculated according to each hidden state:

$$h = W_o \cdot h_1 + W_o \cdot h_2 + \cdots + W_o \cdot h_t + b_o. \tag{60}$$

For simplicity, we replace it with $h = W_o \cdot h_t + b_o$. The prediction of the input X is $y = \arg\max_{c \in [C]} h_c$, where h_c is the c-th component of h.

We mainly consider two kinds of RNN models in this paper: long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) models and gated recurrent unit (GRU) (Cho et al., 2014) models. Each kind of model updates its hidden states differently.

LSTM models update h_t and c_t according to

$$i_t = \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}), \tag{61}$$
$$f_t = \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}), \tag{62}$$
$$g_t = \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}), \tag{63}$$
$$o_t = \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}), \tag{64}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \tag{65}$$
$$h_t = o_t \odot \tanh(c_t), \tag{66}$$

where $\odot$ is the Hadamard product.

GRU models update h_t according to

$$r_t = \sigma(W_{xr}x_t + b_{xr} + W_{hr}h_{t-1} + b_{hr}), \tag{67}$$
$$z_t = \sigma(W_{xz}x_t + b_{xz} + W_{hz}h_{t-1} + b_{hz}), \tag{68}$$
$$n_t = \tanh(W_{xn}x_t + b_{xn} + r_t \odot (W_{hn}h_{t-1} + b_{hn})), \tag{69}$$
$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1}. \tag{70}$$

E Additional Experiment Setup

Datasets. We use the following benchmark datasets for evaluation:

MNIST dataset (LeCun et al., 1998), which is a is a large database of handwritten digits that is commonly used for training various image processing systems. The MNIST dataset contains 60,000 training images and 10,000 testing images.

Rotten Tomatoes Movie Review dataset (Pang and Lee, 2005), which is a benchmark corpus of movie reviews used for sentiment analysis. The RT dataset contains about 39000 and 4800 samples in the training set and the testing set, respectively.

Stanford Sentiment Treebank dataset (Socher et al., 2013), which is also a benchmark corpus of movie reviews used for sentiment analysis. The SST dataset contains about 67349 and 1821 samples in the training set and the testing set, respectively.

Yelp Reviews Polarity dataset (Zhang et al., 2015), which is a large text classification benchmark. The Yelp dataset contains about 560000 and 38000 samples in the training set and the testing set, respectively.

Models. We use the following models for evaluation:

Convolutional Neural Network models, which are classic models for image classification. The CNNs used in our experiments are:

- CNN1: 2 conv layer, 2 linear layer, first linear layer size 392*128 width.
- CNN2: 2 conv layer, 2 linear layer, first linear layer size 784*128 width.
- CNN3: 4 conv layer, 3 linear layer, first linear layer size 392*256 width.
- CNN4: 4 conv layer, 3 linear layer, first linear layer size 784*256 width.
- CNN5: 4 conv layer, 3 linear layer, first linear layer size 392*512 width.
- CNN6: 4 conv layer, 3 linear layer, first linear layer size 784*512 width.
- CNN7: 3 conv layer, 2 linear layer, first linear layer size 392*64 width.
- CNN8: 3 conv layer, 2 linear layer, first linear layer size 784*128 width.
- CNN9: 3 conv layer, 2 linear layer, first linear layer size 288*64 width.
- CNN10: 3 conv layer, 2 linear layer, first linear layer size 576*128 width.

Recurrent Neural Network models, including LSTM (Hochreiter and Schmidhuber, 1997), and GRU (Cho et al., 2014), are classic models for NLP tasks. We use RNNs that consist of 32 and 64 hidden neurons (consistent with those used in previous work (Du et al., 2021)). We refer to them as LSTM-32 (L-32), LSTM-64 (L-64), GRU-32 (G-32), and GRU-64 (G-64).

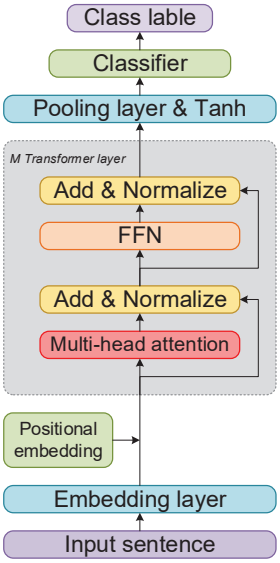


Figure 7: The Transformers used in this work.

Transformer models are the most widely applied models for NLP tasks. We use a Transformer with 3 layers, 4 attention heads, 128 hidden neurons (the feed-forward network in it also has 128 hidden neurons), as shown in Fig. 7.

Baseline. We choose the following methods as baselines.

CROWN (Zhang et al., 2018) achieves the highest precision on almost all models in the evaluation [1] of a recent survey (Li et al., 2023), which is also part of the `auto_LiRPA` library (Xu et al., 2020).

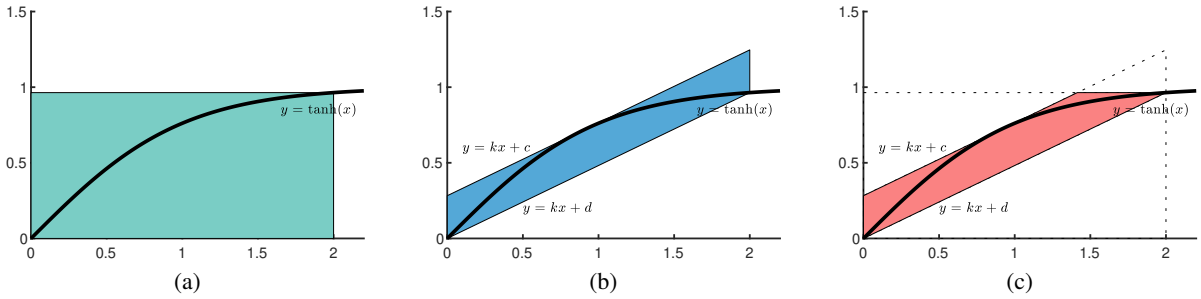
CROWN-BaF (Shi et al., 2020) is the first method proposed for certifying Transformers, which uses the LiRPA procedure to bound the output scores (it is part the of `auto_LiRPA` library).

DeepT (Bonaert et al., 2021) is a zonotope-based method, which surpasses CROWN-BaF in precision at the cost of longer running time.

GaLileo (Zhang et al., 2024) also uses the LiRPA procedure but adopts a more precise relaxation of softmax functions. GaLileo focuses on tightening the relaxation of the self-attention mechanism (the softmax in it) while this work focuses on tightening the relaxation of activation functions.

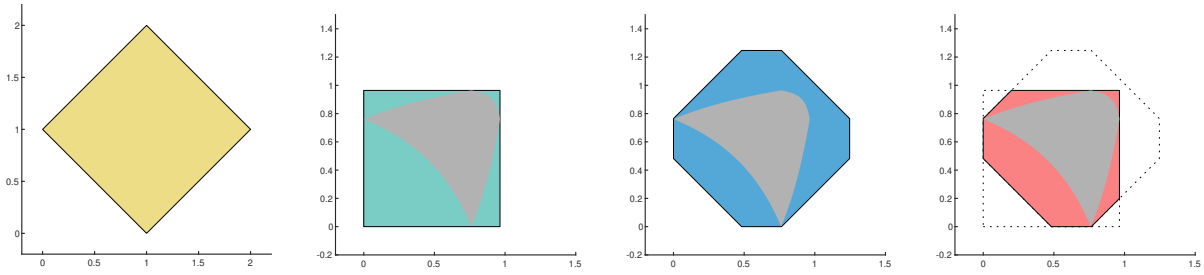
Hardware. Experiments on Transformers are conducted on a server with an Intel Core i9-10920X CPU running at 3.50 GHz, 128 GB memory, and a GeForce RTX 3090 GPU card. Others are conducted on a server with an Intel Xeon Silver 4210R CPU running at 2.40GHz, 128 GB memory, and a GeForce RTX 2080Ti GPU card.

[1] https://sokcertifiedrobustness.github.io/benchmark/



(a) (b) (c)

Figure 8: Extension of DROWN to tanh and zonotopes: the box-style relaxation used in IBP (Gowal et al., 2019) (a), the parallelogram-style relaxation used in Cert-RNN (Du et al., 2021) (b), and their intersection.

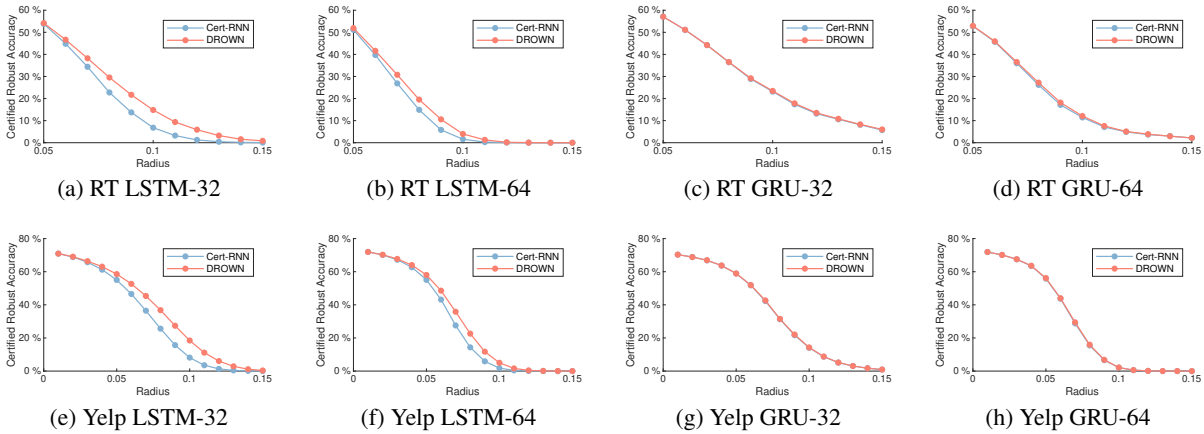


(a) (b) (c) (d)

Figure 9: An intuitive example of different relaxations' over-approximation under tanh. (a) the input set $\mathcal{I}_0$, (b) the box-style relaxation area $\mathcal{O}_1$, (c) the parallelogram-style relaxation area $\mathcal{O}_2$, (d) the refined relaxation area $\mathcal{O}_1 \bigcap \mathcal{O}_2$. The yellow area is the input set and the grey area $\mathcal{O}_0$ is its exact image under tanh.

Dataset		RT				Yelp			
Model		L-32	L-64	G-32	G-64	L-32	L-64	G-32	G-64
Clean Acc.		73.78%	73.63%	73.49%	74.71%	72.39%	73.16%	71.34%	73.14%
C.Acc.	Base	6.80%	5.79%	17.38%	17.14%	15.67%	14.32%	21.69%	28.81%
	Ours	**14.81%**	**10.58%**	**17.83%**	**18.18%**	**27.37%**	**22.60%**	**22.07%**	**29.44%**
Time(s)	Base	73.8	73.5	73.5	74.5	9.1e2	9.8e2	4.5e2	5.4e2
	Ours	73.7	73.6	73.6	74.5	1.1e3	1.1e3	5.3e2	6.4e2

Table 6: Evaluation results of Cert-RNN (Base) and DROWN (Ours) on the RNN models against multi-word attacks.



(a) RT LSTM-32 (b) RT LSTM-64 (c) RT GRU-32 (d) RT GRU-64

(e) Yelp LSTM-32 (f) Yelp LSTM-64 (g) Yelp GRU-32 (h) Yelp GRU-64

Figure 10: Certified robust accuracy of the four models on the RT and Yelp dataset at different ε_e computed using the baseline method (Cert-RNN) and DROWN.

F Evaluations of Extending DROWN to Zonotope-based Methods on RNNs with Tanh Activation

In this subsection, we present results of our early research on extending DROWN to zonotope-based methods, which is evaluated on RNNs with tanh activation rather than ReLU. Zonotope-based methods formulate the adversarial space $\mathcal{S}(x_0)$ as a zonotope and propagate it through the model to derive lower and upper bounds for each score.

We briefly introduce our relaxation for tanh and zonotopes. In contrast to those for LiRPA-based methods, relaxations for zonotopes require parallel bounds, i.e, the slopes of the lower and upper bounds need to be identical. As shown in Fig. 8, the first pair of linear bounds are the lower and upper edges of smallest box containing the tanh curve (i.e., the box-style relaxation used in IBP) and the second pair of linear bounds are the lower and upper edges of smallest parallelogram containing the curve (i.e., the parallelogram-style relaxation used in Cert-RNN). The intersection of the relaxation areas of the above relaxations is smaller, which indicates combining the two relaxations can tighten tanh's relaxation. We also provide an intuitive example in Fig. 9. Given an input set $\mathcal{I}_0$, we denote the exact image of $\mathcal{I}_0$ under tanh as $\mathcal{O}_0$. We calculate an over-approximation of $\mathcal{O}_0$ using the box-style relaxation (denoted as $\mathcal{O}_1$) and another over-approximation of $\mathcal{O}_0$ using the parallelogram-style relaxation (denoted as $\mathcal{O}_2$). As shown in Fig. 9, the refined relaxation area $\mathcal{O}_1 \bigcap \mathcal{O}_2$ is tighter.

We evaluate the precision and efficiency of the baseline method Cert-RNN (Du et al., 2021) and ours against the worst-case adversarial attack where all words in the sentence can be perturbed (under the ℓ_∞ norm). We report the results in Tab. 6 and Fig. 10.

Precision. As shown in Tab. 6, DROWN achieves higher certified robust accuracy. For example, for the LSTM-32 model on the RT dataset, the baseline proves that 6.80% samples are robust while DROWN proves that 14.81% samples are robust. We further report the certified robust accuracy within ℓ_p-balls with different radius ε in Fig. 10. Both of them show that the certified robust accuracy computed by DROWN is higher than the baseline, which indicates that the intersection-based relaxation is more precise than the classic one. Moreover, we observe that the precision improvement of DROWN is more obvious in LSTM models than in GRU models. This is because LSTM models involve more variables and use more non-linear functions and the advantage of DROWN comes from the relaxation of non-linear functions. Thus, DROWN is more precise, especially on complex models with more non-linear functions.

Efficiency. In addition to state-of-the-art certified robust accuracy achieved by DROWN, we can see from Tab. 6 that the running times of DROWN are close to those of the baseline. The reason is that the computational cost of the box-style relaxation is negligible compared to those of the parallelogram-style relaxation. For example, for the LSTM-64 model on the RT dataset, DROWN takes 73.5 seconds to certify the robustness of all test samples while the baseline takes 73.6 seconds. Though DROWN is slower, we argue that it is acceptable because the gap between the two methods is far smaller than the running times of the baseline.

In conclusion, we believe that DROWN is a better choice for certifying the robustness of RNN models due to its superiority in precision and adequate efficiency.