ROBUSTNESS VERIFICATION FOR TRANSFORMERS

Zhouxing Shi[1], **Huan Zhang**[2], **Kai-Wei Chang**[2], **Minlie Huang**[1], **Cho-Jui Hsieh**[2]
[1]Dept. of Computer Science & Technology, Tsinghua University, Beijing 10084, China
[2]Dept. of Computer Science, University of California, Los Angeles, CA 90095, USA
zhouxingshichn@gmail.com, huan@huan-zhang.com
kw@kwchang.net, aihuang@tsinghua.edu.cn, chohsieh@cs.ucla.edu

ABSTRACT

Robustness verification that aims to formally certify the prediction behavior of neural networks has become an important tool for understanding the behavior of a given model and for obtaining safety guarantees. However, previous methods are usually limited to relatively simple neural networks. In this paper, we consider the robustness verification problem for Transformers. Transformers have complex self-attention layers that pose many challenges for verification, including cross-nonlinearity and cross-position dependency, which have not been discussed in previous work. We resolve these challenges and develop the first verification algorithm for Transformers. The certified robustness bounds computed by our method are significantly tighter than those by naive Interval Bound Propagation. These bounds also shed light on interpreting Transformers as they consistently reflect the importance of words in sentiment analysis.

1 INTRODUCTION

Deep neural networks have been successfully applied to many domains. However, a major criticism is that these black box models are difficult to analyze and their behavior is not guaranteed. Moreover, it has been shown that the predictions of deep networks become unreliable and unstable when tested in unseen situations, e.g., in the presence of small and adversarial perturbation to the input (Szegedy et al., 2013; Goodfellow et al., 2014; Lin et al., 2019). Therefore, neural network verification has become an important tool for analyzing and understanding the behavior of neural networks, with applications in safety-critical applications (Katz et al., 2017; Julian et al., 2019; Lin et al., 2019), model explanation (Shih et al., 2018) and robustness analysis (Tjeng et al., 2019; Wang et al., 2018c; Gehr et al., 2018; Wong & Kolter, 2018; Singh et al., 2018; Weng et al., 2018; Zhang et al., 2018).

Formally, a neural network verification algorithm aims to provably characterize the prediction of a network within some input space. For example, given a K-way classification model $f : \mathbb{R}^d \to \mathbb{R}^K$, we can verify some linear specification (defined by a vector c) as below:

$$\min_{\mathbf{x}} \sum_i c_i f_i(\mathbf{x}) \quad \text{s.t. } \mathbf{x} \in S, \tag{1}$$

where S is a predefined input space. For example, in the **robustness verification problem** that we are going to focus on in this paper, $S = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_0\|_p \leq \epsilon\}$ is defined as some small ℓ_p-ball around the original example $\mathbf{x}_0$, and setting up $c = 1_{y_0} - 1_y$ can verify whether the logit output of class y_0 is always greater than another class y within S. This is a nonconvex optimization problem which makes computing the exact solution challenging, and thus algorithms are recently proposed to find lower bounds of Eq. (1) in order to efficiently obtain a safety guarantee (Gehr et al., 2018; Weng et al., 2018; Zhang et al., 2018; Singh et al., 2019). Moreover, extension of these algorithms can be used for verifying some properties beyond robustness, such as rotation or shift invariant (Singh et al., 2019), conservation of energy (Qin et al., 2019) and model correctness (Yang & Rinard, 2019).

However, most of existing verification methods focus on relatively simple neural network architectures, such as feed-forward and recurrent neural networks, and cannot handle complex structures. In this paper, we develop the first robustness verification algorithm for Transformers (Vaswani et al., 2017) with self-attention layers. Transformers have been widely used in natural language processing (Devlin et al., 2018; Yang et al., 2019; Liu et al., 2019) and many other domains (Parmar et al.,

2018; Kang & McAuley, 2018; Li et al., 2019b; Su et al., 2019; Li et al., 2019a). For frames under perturbation in the input sequence, we aim to compute a lower bound ϵ such that when these frames are perturbed within ℓ_p-balls centered at the original frames respectively and with a radius of ϵ, the model prediction is certified to be unchanged. To compute such a bound efficiently, we adopt the linear-relaxation framework (Weng et al., 2018; Zhang et al., 2018) – we recursively propagate and compute linear lower bound and upper bound for each neuron with respect to the input within perturbation set S.

We resolve several particular challenges in verifying Transformers. **First**, Transformers with self-attention layers have a complicated architecture. Unlike simpler networks, they cannot be written as multiple layers of linear transformations or element-wise operations. Therefore, we need to propagate linear bounds differently for self-attention layers. **Second**, dot products, softmax, and weighted summation in self-attention layers involve multiplication or division of two variables under perturbation, namely cross-nonlinearity, which is not present in feed-forward networks. Ko et al. (2019) proposed a gradient descent based approach to find linear bounds, however it is inefficient and poses a computational challenge for transformer verification as self-attention is the core of transformers. In contrast, we derive closed-form linear bounds that can be computed in $O(1)$ complexity. **Third**, neurons in each position after a self-attention layer depend on *all* neurons in different positions before the self-attention (namely *cross-position dependency*), unlike the case in recurrent neural networks where outputs depend on only the hidden features from the previous position and the current input. Previous works (Zhang et al., 2018; Weng et al., 2018; Ko et al., 2019) have to track all such dependency and thus is costly in time and memory. To tackle this, we introduce an efficient bound propagating process in a forward manner specially for self-attention layers, enabling the tighter backward bounding process for other layers to utilize bounds computed by the forward process. In this way, we avoid cross-position dependency in the backward process which is relatively slower but produces tighter bounds. Combined with the forward process, the complexity of the backward process is reduced by $O(n)$ for input length n, while the computed bounds remain comparably tight. Our contributions are summarized below:

- We propose an effective and efficient algorithm for verifying the robustness of Transformers with self-attention layers. To our best knowledge, this is the first method for verifying Transformers.
- We resolve key challenges in verifying Transformers, including cross-nonlinearity and cross-position dependency. Our bounds are significantly tighter than those by adapting Interval Bound Propagation (IBP) (Mirman et al., 2018; Gowal et al., 2018).
- We quantitatively and qualitatively show that the certified lower bounds consistently reflect the importance of input words in sentiment analysis, which justifies that the computed bounds are meaningful in practice.

2 Related Work

Robustness Verification for Neural Networks. Given an input $\mathbf{x}_0$ and a small region $\mathbb{B}_p(\mathbf{x}_0, \epsilon) := \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_0\|_p \leq \epsilon\}$, the goal of robustness verification is to verify whether the prediction of the neural network is unchanged within this region. This problem can be mathematically formulated as Eq. (1). If Eq. (1) can be solved optimally, then we can derive the minimum adversarial perturbation of $\mathbf{x}$ by conducting binary search on ϵ. Equivalently, we obtain the maximum ϵ such that any perturbation within $\mathbb{B}_p(\mathbf{x}_0, \epsilon)$ cannot change the predicted label.

Several works focus on solving Eq. (1) exactly and optimally, using mixed integer linear programming (MILP) (Tjeng et al., 2019; Dutta et al., 2018), branch and bound (BaB) (Bunel et al., 2018), and satisfiability modulo theory (SMT) (Ehlers, 2017; Katz et al., 2017). Unfortunately, due to the nonconvexity of model f, solving Eq. (1) is NP-hard even for a simple ReLU network (Katz et al., 2017). Therefore, we can only hope to compute a lower bound of Eq. (1) efficiently by using relaxations. Many algorithms can be seen as using convex relaxations for non-linear activation functions (Salman et al., 2019), including using duality (Wong & Kolter, 2018; Dvijotham et al., 2018), abstract domains (Gehr et al., 2018; Singh et al., 2018; Mirman et al., 2018; Singh et al., 2019), layer-by-layer reachability analysis (Wang et al., 2018b; Weng et al., 2018; Zhang et al., 2018; Gowal et al., 2018) and semi-definite relaxations (Raghunathan et al., 2018; Dvijotham et al., 2019). Additionally, robustness verification can rely on analysis on local Lipschitz constants (Hein & Andriushchenko, 2017; Zhang et al., 2019). However, existing methods are mostly limited to verifying

networks with relatively simple architectures, such as feed-forward networks and RNNs (Wang et al., 2018a; Akintunde et al., 2019; Ko et al., 2019), while none of them are able to handle Transformers.

Transformers and Self-Attentive Models. Transformers (Vaswani et al., 2017) based on the self-attention mechanism, further with pre-training on large-scale corpora, such as BERT (Devlin et al., 2018), XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019), achieved state-of-the-art performance on many NLP tasks. Self-attentive models are also useful beyond NLP, including VisualBERT for extracting features from both text and images (Li et al., 2019b; Su et al., 2019), image transformer for image generation (Parmar et al., 2018), acoustic models for speech recognition, sequential recommendation (Kang & McAuley, 2018) and graph embedding (Li et al., 2019a).

The robustness of NLP models has been studied, especially many methods have been proposed to generate adversarial examples (Papernot et al., 2016; Jia & Liang, 2017; Zhao et al., 2017; Alzantot et al., 2018; Cheng et al., 2018; Ebrahimi et al., 2018). In particular, Hsieh et al. (2019) showed that Transformers are more robust than LSTMs. However, there is not much work on robustness verification for NLP models. Ko et al. (2019) verified RNN/LSTM. Jia et al. (2019); Huang et al. (2019) used Interval Bound Propagation (IBP) for certified robustness training of CNN and LSTM. In this paper, we propose the first verification method for Transformers.

3 METHODOLOGY

We aim to verify the robustness of a Transformer whose input is a sequence of frames $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(n)}]$. We take binary text classification as an running example, where $\mathbf{x}^{(i)}$ is a word embedding and the model outputs a score $y_c(\mathbf{X})$ for each class c ($c \in \{0, 1\}$). Nevertheless, our method for verifying Transformers is general and can also be applied in other applications.

For a clean input sequence $\mathbf{X}_0 = [\mathbf{x}_0^{(1)}, \mathbf{x}_0^{(2)}, \cdots, \mathbf{x}_0^{(n)}]$ correctly classified by the model, let $P = \{r_1, r_2, \cdots, r_t\} (1 \leq r_k \leq n)$ be the set of perturbed positions, where t is the number of perturbed positions. Thus the perturbed input will belong to $\mathbb{S}_\epsilon := \{\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(n)}] : \|\mathbf{x}^{(r_k)} - \mathbf{x}_0^{(r_k)}\|_p \leq \epsilon, 1 \leq k \leq t, \mathbf{x}^{(i)} = \mathbf{x}_0^{(i)}, \forall i \notin P\}$. Assuming that c is the gold class, the goal of robustness verification is to compute

$$\{\min_{\mathbf{X} \in \mathbb{S}} y_c(\mathbf{X}) - y_{1-c}(\mathbf{X})\} := \delta_\epsilon(\mathbf{X}).$$

If $\delta_\epsilon(\mathbf{X}) > 0$, the output score of the correct class will always be larger than the incorrect one within $\mathbb{S}_\epsilon$. As mentioned previously, computing the exact values of $\delta_\epsilon(\mathbf{X})$ is NP-hard, and thus our goal is to efficiently compute a lower bound $\delta_\epsilon^L(\mathbf{X}) \leq \delta_\epsilon(\mathbf{X})$.

3.1 BASE FRAMEWORK

We obtain $\delta_\epsilon^L(\mathbf{X})$ by computing the bounds of each neuron when $\mathbf{X}$ is perturbed within $\mathbb{S}_\epsilon$ (δ_ϵ^L can be regarded as a final neuron). A Transformer layer can be decomposed into a number of sub-layers, where each sub-layer contains neurons after some operation. These operations can be categorized into three categories: 1) linear transformations, 2) unary nonlinear functions, and 3) operations in self-attention. Each sub-layer contains n positions in the sequence and each position contains a group of neurons. We assume that the Transformer we verify has m sub-layers in total, and the value of the j-th neuron at the i-th position in the l-th sub-layer is $\Phi_j^{(l,i)}(\mathbf{X})$, where $\Phi^{(l,i)}(\mathbf{X})$ is a vector for the specified sub-layer and position. Specially, $\Phi^{(0,i)} = \mathbf{x}^{(i)}$ taking $l = 0$. We aim to compute a global lower bound $f_j^{(l,i),L}$ and a global upper bound $f_j^{(l,i),U}$ of $\Phi_j^{(l,i)}(\mathbf{X})$ for $\mathbf{X} \in \mathbb{S}_\epsilon$.

We compute bounds from the first sub-layer to the last sub-layer. For neurons in the l-th layer, we aim to represent their bounds as linear functions of neurons in a previous layer, the l'-th layer:

$$\sum_{k=1}^{n} \mathbf{\Lambda}_{j,:}^{(l,i,l',k),L} \Phi^{(l',k)}(\mathbf{X}) + \mathbf{\Delta}_j^{(l,i,l'),L} \leq \Phi_j^{(l,i)}(\mathbf{X}) \leq \sum_{k=1}^{n} \mathbf{\Lambda}_{j,:}^{(l,i,l',k),U} \Phi^{(l',k)}(\mathbf{X}) + \mathbf{\Delta}_j^{(l,i,l'),U}, \quad (2)$$

where $\mathbf{\Lambda}^{(l,i,l',k),L}, \mathbf{\Delta}^{(l,i,l'),L}$ and $\mathbf{\Lambda}^{(l,i,l',k),U}, \mathbf{\Delta}^{(l,i,l'),U}$ are parameters of linear lower and upper bounds respectively. Using linear bounds enables us to efficiently compute bounds with a reasonable tightness. We initially have $\mathbf{\Lambda}^{(l,i,l,i),L} = \mathbf{\Lambda}^{(l,i,l,i),U} = \mathbf{I}$ and $\mathbf{\Delta}^{(l,i,l),L} = \mathbf{\Delta}^{(l,i,l),U} = \mathbf{0}$. Thereby the right-hand-side of Eq. (2) equals to $\Phi_j^{(l,i)}(\mathbf{X})$ when $l' = l$. Generally, we use a backward

process to propagate the bounds to previous sub-layers, by substituting $\Phi^{(l',i)}$ with linear functions of previous neurons. It can be recursively conducted until the input layer $l' = 0$. Since $\Phi^{(0,k)} = \mathbf{x}^{(k)} = \mathbf{x}_0^{(k)} (\forall k \notin P)$ is constant, we can regard the bounds as linear functions of the perturbed embeddings $\Phi^{(0,r_k)} = \mathbf{x}^{(r_k)} (1 \leq k \leq t)$, and take the global bounds for $\mathbf{x}^{(r_k)} \in \mathbb{B}_p(\mathbf{x}_0^{(r_k)}, \epsilon)$:

$$f_j^{(l,i),L} = -\epsilon \sum\nolimits_{k=1}^{t} \parallel \mathbf{\Lambda}_{j,:}^{(l,i,0,r_k),L} \parallel_q + \sum\nolimits_{k=1}^{n} \mathbf{\Lambda}_{j,:}^{(l,i,0,k),L} \mathbf{x}_0^{(k)} + \mathbf{\Delta}_j^{(l,i,0),L}, \tag{3}$$

$$f_j^{(l,i),U} = \epsilon \sum\nolimits_{k=1}^{t} \parallel \mathbf{\Lambda}_{j,:}^{(l,i,0,r_k),U} \parallel_q + \sum\nolimits_{k=1}^{n} \mathbf{\Lambda}_{j,:}^{(l,i,0,k),U} \mathbf{x}_0^{(k)} + \mathbf{\Delta}_j^{(l,i,0),U}, \tag{4}$$

where $1/p + 1/q = 1$ with $p, q \geq 1$. These steps resemble to CROWN (Zhang et al., 2018) which is proposed to verify feed-forward networks. We further support verifying self-attentive Transformers which are more complex than feed-forward networks. Moreover, unlike CROWN that conducts a full backward process, we combine the backward process with a forward process (see Sec. 3.3) to reduce the computational complexity of verifying Transformers.

3.2 Linear Transformations and Unary Nonlinear Functions

Linear transformations and unary nonlinear functions are basic operations in neural networks. We show how bounds Eq. (2) at the l'-th sub-layer are propagated to the $(l'-1)$-th layer.

Linear Transformations If the l'-th sub-layer is connected with the $(l'-1)$-th sub-layer with a linear transformation $\Phi^{(l',k)}(\mathbf{X}) = \mathbf{W}^{(l')} \Phi^{(l'-1,k)}(\mathbf{X}) + \mathbf{b}^{(l')}$ where $\mathbf{W}^{(l')}, \mathbf{b}^{(l')}$ are parameters of the linear transformation, we propagate the bounds to the $(l'-1)$-th layer by substituting $\Phi^{(l',k)}(\mathbf{X})$:

$$\mathbf{\Lambda}^{(l,i,l'-1,k),L/U} = \mathbf{\Lambda}^{(l,i,l',k),L/U} \mathbf{W}^{(l')}, \mathbf{\Delta}^{(l,i,l'-1),L/U} = \mathbf{\Delta}^{(l,i,l'),L/U} + \left(\sum\nolimits_{k=1}^{n} \mathbf{\Lambda}^{(l,i,l',k),L/U} \right) \mathbf{b}^{(l')},$$

where "L/U" means that the equations hold for both lower bounds and upper bounds respectively.

Unary Nonlinear Functions If the l'-th layer is obtained from the $(l'-1)$-th layer with an unary nonlinear function $\Phi_j^{(l',k)}(\mathbf{X}) = \sigma^{(l')}(\Phi_j^{(l'-1,k)}(\mathbf{X}))$, to propagate linear bounds over the nonlinear function, we first bound $\sigma^{(l')}(\Phi_j^{(l'-1,k)}(\mathbf{X}))$ with two linear functions of $\Phi_j^{(l'-1,k)}(\mathbf{X})$:

$$\alpha_j^{(l',k),L} \Phi_j^{(l'-1,k)}(\mathbf{X}) + \beta_j^{(l',k),L} \leq \sigma^{(l')}(\Phi_j^{(l'-1,k)}(\mathbf{X})) \leq \alpha_j^{(l',k),U} \Phi_j^{(l'-1,k)}(\mathbf{X}) + \beta_j^{(l',k),U},$$

where $\alpha_j^{(l',k),L/U}, \beta_j^{(l',k),L/U}$ are parameters such that the inequation holds for all $\Phi_j^{(l'-1,k)}(\mathbf{X})$ within its bounds computed previously. Such linear relaxations can be done for different functions, respectively. We provide detailed bounds for functions involved in Transformers in Appendix B.

We then back propagate the bounds:

$$\mathbf{\Lambda}_{:,j}^{(l,i,l'-1,k),L/U} = \alpha_j^{(l',k),L/U} \mathbf{\Lambda}_{:,j,+}^{(l,i,l',k),L/U} + \alpha_j^{(l',k),U/L} \mathbf{\Lambda}_{:,j,-}^{(l,i,l',k),L/U},$$

$$\mathbf{\Delta}_j^{(l,i,l'-1),L/U} = \mathbf{\Delta}_j^{(l,i,l'),L/U} + \left(\sum\nolimits_{k=1}^{n} \beta_j^{(l',k),L/U} \mathbf{\Lambda}_{:,j,+}^{(l,i,l',k),L/U} + \beta_j^{(l',k),U/L} \mathbf{\Lambda}_{:,j,-}^{(l,i,l',k),L/U} \right),$$

where $\mathbf{\Lambda}_{:,j,+}^{(l,i,l',k),L/U}$ and $\mathbf{\Lambda}_{:,j,-}^{(l,i,l',k),L/U}$ mean to retain positive and negative elements in vector $\mathbf{\Lambda}_{:,j}^{(l,i,l',k),L/U}$ respectively and set other elements to 0.

3.3 Self-Attention Mechanism

Self-attention layers are the most challenging parts for verifying Transformers. We assume that $\Phi^{(l-1,i)}(\mathbf{X})$ is the input to a self-attention layer. We describe our method for computing bounds for one attention head, and bounds for different heads of the multi-head attention in Transformers can be easily concatenated. $\Phi^{(l-1,i)}(\mathbf{X})$ is first linearly projected to queries $\mathbf{q}^{(l,i)}(\mathbf{X})$, keys $\mathbf{k}^{(l,i)}(\mathbf{X})$, and values $\mathbf{v}^{(l,i)}(\mathbf{X})$ with different linear projections, and their bounds can be obtained as described in Sec. 3.2. We also keep their linear bounds that are linear functions of the perturbed embeddings.

For convenience, let $\mathbf{x}^{(r)} = \mathbf{x}^{(r_1)} \oplus \mathbf{x}^{(r_2)} \oplus \cdots \mathbf{x}^{(r_t)}$, where $\oplus$ indicates vector concatenation, and thereby we represent the linear bounds as linear functions of $\mathbf{x}^{(r)}$:

$$\mathbf{\Omega}_{j,:}^{(l,i),q/k/v,L}\mathbf{x}^{(r)} + \mathbf{\Theta}_{j}^{(l,i),q/k/v,L} \leq (\mathbf{q}/\mathbf{k}/\mathbf{v})_{j}^{(l,i)}(\mathbf{X}) \leq \mathbf{\Omega}_{j,:}^{(l,i),q/k/v,U}\mathbf{x}^{(r)} + \mathbf{\Theta}_{j}^{(l,i),q/k/v,U},$$

where $q/k/v$ and $\mathbf{q}/\mathbf{k}/\mathbf{v}$ mean that the inequation holds for queries, keys and values respectively. We then bound the output of the self-attention layer starting from $\mathbf{q}^{(l,i)}(\mathbf{X})$, $\mathbf{k}^{(l,i)}(\mathbf{X})$, $\mathbf{v}^{(l,i)}(\mathbf{X})$.

Bounds of Multiplications and Divisions We bound multiplications and divisions in the self-attention mechanism with linear functions. We aim to bound bivariate function $z = xy$ or $z = \frac{x}{y}(y > 0)$ with two linear functions $z^L = \alpha^L x + \beta^L y + \gamma^L$ and $z^U = \alpha^U x + \beta^U y + \gamma^U$, where $x \in [l_x, u_x], y \in [l_y, u_y]$ are bounds of x, y obtained previously. For $z = xy$, we derive optimal parameters: $\alpha^L = l_y$, $\alpha^U = u_y$, $\beta^L = \beta^U = l_x$, $\gamma^L = -l_x l_y$, $\gamma^U = -l_x u_y$. We provide a proof in Appendix C. However, directly bounding $z = \frac{x}{y}$ is tricky; fortunately, we can bound it indirectly by first bounding a unary function $\overline{y} = \frac{1}{y}$ and then bounding the multiplication $z = x\overline{y}$.

A Forward Process For the self-attention mechanism, instead of using the backward process like CROWN (Zhang et al., 2018), we compute bounds with a forward process which we will show later that it can reduce the computational complexity. Attention scores are computed from $\mathbf{q}^{(l,i)}(\mathbf{X})$ and $\mathbf{k}^{(l,i)}(\mathbf{X})$: $\mathbf{S}_{i,j}^{(l)} = (\mathbf{q}^{(l,i)}(\mathbf{X}))^T \mathbf{k}^{(l,j)}(\mathbf{X}) = \sum_{k=1}^{d_{qk}} \mathbf{q}_k^{(l,i)}(\mathbf{X})\mathbf{k}_k^{(l,j)}(\mathbf{X})$, where d_{qk} is the dimension of $\mathbf{q}^{(l,i)}(\mathbf{X})$ and $\mathbf{k}^{(l,j)}(\mathbf{X})$. For each multiplication $\mathbf{q}_k^{(l,i)}(\mathbf{X})\mathbf{k}_k^{(l,j)}(\mathbf{X})$, it is bounded by:

$$\mathbf{q}_k^{(l,i)}(\mathbf{X})\mathbf{k}_k^{(l,j)}(\mathbf{X}) \geq \alpha_k^{(l,i,j),L}\mathbf{q}_k^{(l,i)}(\mathbf{X}) + \beta_k^{(l,i,j),L}\mathbf{k}_k^{(l,j)}(\mathbf{X}) + \gamma_k^{(l,i,j),L},$$
$$\mathbf{q}_k^{(l,i)}(\mathbf{X})\mathbf{k}_k^{(l,j)}(\mathbf{X}) \leq \alpha_k^{(l,i,j),U}\mathbf{q}_k^{(l,i)}(\mathbf{X}) + \beta_k^{(l,i,j),U}\mathbf{k}_k^{(l,j)}(\mathbf{X}) + \gamma_k^{(l,i,j),U}.$$

We then obtain the bounds of $\mathbf{S}_{i,j}^{(l)}$:

$$\mathbf{\Omega}_{j,:}^{(l,i),s,L}\mathbf{x}^{(r)} + \mathbf{\Theta}_{j}^{(l,i),s,L} \leq \mathbf{S}_{i,j}^{(l)} \leq \mathbf{\Omega}_{j,:}^{(l,i),s,U}\mathbf{x}^{(r)} + \mathbf{\Theta}_{j}^{(l,i),s,U},$$

$$\begin{aligned}
\mathbf{\Omega}_{j,:}^{(l,i),s,L/U} &= \alpha_k^{(l,i,j),L/U}\Big(\sum\nolimits_{\alpha_k^{(l,i,j),L/U}>0} \mathbf{\Omega}_{k,:}^{(l,i),q,L/U} + \sum\nolimits_{\alpha_k^{(l,i,j),L/U}<0} \mathbf{\Omega}_{k,:}^{(l,i),q,U/L}\Big) + \\
&\quad \beta_k^{(l,i,j),L/U}\Big(\sum\nolimits_{\beta_k^{(l,i,j),L/U}>0} \mathbf{\Omega}_{k,:}^{(l,j),k,L/U} + \sum\nolimits_{\beta_k^{(l,i,j),L/U}<0} \mathbf{\Omega}_{k,:}^{(l,j),k,U/L}\Big), \\
\mathbf{\Theta}_{j}^{(l,i),s,L/U} &= \alpha_k^{(l,i,j),L/U}\Big(\sum\nolimits_{\alpha_k^{(l,i,j),L/U}>0} \mathbf{\Theta}_{k}^{(l,i),q,L/U} + \sum\nolimits_{\alpha_k^{(l,i,j),L/U}<0} \mathbf{\Theta}_{k}^{(l,i),q,U/L}\Big) + \\
&\beta_k^{(l,i,j),L/U}\Big(\sum\nolimits_{\beta_k^{(l,i,j),L/U}>0} \mathbf{\Theta}_{k}^{(l,j),k,L/U} + \sum\nolimits_{\beta_k^{(l,i,j),L/U}<0} \mathbf{\Theta}_{k}^{(l,j),k,U/L}\Big) + \sum\nolimits_{k=1}^{d_{qk}} \gamma_k^{(l,i,j),L/U}.
\end{aligned}$$

In this way, linear bounds of $\mathbf{q}^{(l,i)}(\mathbf{X})$ and $\mathbf{k}^{(l,i)}(\mathbf{X})$ are forward propagated to $\mathbf{S}_{i,j}^{(l)}$. Attention scores are normalized into attention probabilities with a softmax, i.e. $\tilde{\mathbf{S}}_{i,j}^{(l)} = \exp(\mathbf{S}_{i,j}^{(l)})/(\sum_{k=1}^{n}\exp(\mathbf{S}_{i,k}^{(l)}))$, where $\tilde{\mathbf{S}}_{i,j}^{(l)}$ is a normalized attention probability. $\exp(\mathbf{S}_{i,j}^{(l)})$ is an unary nonlinear function and can be bounded by $\alpha_{i,j}^{(l),L/U}\mathbf{S}_{i,j}^{(l)} + \beta_{i,j}^{(l),L/U}$. So we forward propagate bounds of $\mathbf{S}_{i,j}^{(l)}$ to bound $\exp(\mathbf{S}_{i,j}^{(l)})$ with $\mathbf{\Omega}_{j,:}^{(l,i),e,L/U}\mathbf{x}^{(r)} + \mathbf{\Theta}_{j}^{(l,i),e,L/U}$, where:

$$\begin{cases}
\mathbf{\Omega}_{j,:}^{(l,i),e,L/U} = \alpha_{i,j}^{(l),L/U}\mathbf{\Omega}_{j,:}^{(l,i),s,L/U} & \mathbf{\Theta}_{j}^{(l,i),e,L/U} = \alpha_{i,j}^{(l),L/U}\mathbf{\Theta}_{j}^{(l,i),s,L/U} + \beta_{i,j}^{(l),L/U} \quad \alpha_{i,j}^{(l),L/U} \geq 0, \\
\mathbf{\Omega}_{j,:}^{(l,i),e,L/U} = \alpha_{i,j}^{(l),L/U}\mathbf{\Omega}_{j,:}^{(l,i),s,U/L} & \mathbf{\Theta}_{j}^{(l,i),e,L/U} = \alpha_{i,j}^{(l),L/U}\mathbf{\Theta}_{j}^{(l,i),s,U/L} + \beta_{i,j}^{(l),L/U} \quad \alpha_{i,j}^{(l),L/U} < 0.
\end{cases}$$

By summing up bounds of each $\exp(\mathbf{S}_{i,k}^{(l)})$, linear bounds can be further propagated to $\sum_{k=1}^{n}\exp(\mathbf{S}_{i,k}^{(l)})$. With bounds of $\exp(\mathbf{S}_{i,j}^{(l)})$ and $\sum_{k=1}^{n}\exp(\mathbf{S}_{i,k}^{(l)})$ ready, we forward propagate the bounds to $\tilde{\mathbf{S}}_{i,j}^{(l)}$ with a division similarly to bounding $\mathbf{q}_k^{(l,i)}(\mathbf{X})\mathbf{k}_k^{(l,j)}(\mathbf{X})$. The output of the self-attention $\Phi^{(l,i)}(\mathbf{X})$ is obtained with a summation of $\mathbf{v}^{(l,j)}(\mathbf{X})$ weighted by attention probability $\tilde{\mathbf{S}}_{i,k}^{(l)}$: $\Phi_j^{(l,i)}(\mathbf{X}) = \sum_{k=1}^{n}\tilde{\mathbf{S}}_{i,k}^{(l)}\mathbf{v}_j^{(l,k)}(\mathbf{X})$, which can be regarded as a dot product of $\tilde{\mathbf{S}}_{i}^{(l)}$ and $\tilde{\mathbf{v}}_k^{(l,j)}(\mathbf{X})$, where $\tilde{\mathbf{v}}_k^{(l,j)}(\mathbf{X}) = \mathbf{v}_j^{(l,k)}(\mathbf{X})$ whose bounds can be obtained from those of $\mathbf{v}_j^{(l,k)}(\mathbf{X})$

with a transposing. Therefore, bounds of $\tilde{\mathbf{S}}_{i,k}^{(l)}$ and $\tilde{\mathbf{v}}_k^{(l,j)}(\mathbf{X})$ can be forward propagated to $\Phi^{(l,i)}(\mathbf{X})$ similarly to bounding $\mathbf{S}_{i,j}^{(l)}$. In this way, we obtain the output bounds of the self-attention:

$$\mathbf{\Omega}_{j,:}^{(l',i),\Phi,L}\mathbf{x}^{(r)} + \mathbf{\Theta}_j^{(l',i),\Phi,L} \leq \Phi^{(l',i)}(\mathbf{X}) \leq \mathbf{\Omega}_{j,:}^{(l',i),\Phi,U}\mathbf{x}^{(r)} + \mathbf{\Theta}_j^{(l',i),\Phi,U}. \tag{5}$$

Recall that $\mathbf{x}^{(r)}$ is a concatenation of $\mathbf{x}^{(r_1)}, \mathbf{x}^{(r_2)}, \cdots, \mathbf{x}^{(r_t)}$. We can split $\mathbf{\Omega}_{j,:}^{(l',i),\Phi,L/U}$ into t vectors with equal dimensions, $\mathbf{\Omega}_{j,:}^{(l',i,1),\Phi,L/U}, \mathbf{\Omega}_{j,:}^{(l',i,2),\Phi,L/U}, \cdots, \mathbf{\Omega}_{j,:}^{(l',i,t),\Phi,L/U}$, such that Eq. (5) becomes

$$\sum\nolimits_{k=1}^{t} \mathbf{\Omega}_{j,:}^{(l',i,k),\Phi,L}\mathbf{x}^{(r_k)} + \mathbf{\Theta}_j^{(l',i),\Phi,L} \leq \Phi^{(l',i)}(\mathbf{X}) \leq \sum\nolimits_{k=1}^{t} \mathbf{\Omega}_{j,:}^{(l',i,k),\Phi,U}\mathbf{x}^{(r_k)} + \mathbf{\Theta}_j^{(l',i),\Phi,U}. \tag{6}$$

Backward Process to Self-Attention Layers When computing bounds for a later sub-layer, the l-th sub-layer, using the backward process, we directly propagate the bounds at the the closest previous self-attention layer assumed to be the l'-th layer, to the input layer, and we skip other previous sub-layers. The bounds propagated to the l'-th layer are as Eq. (2). We substitute $\Phi^{(l',k)}(\mathbf{X})$ with linear bounds in Eq. (6):

$$\mathbf{\Lambda}_{j,:}^{(l,i,0,r_k),L/U} = \sum\nolimits_{k'=1}^{n} (\mathbf{\Lambda}_{j,:,+}^{(l,i,l',k'),L/U}\mathbf{\Omega}_{j,:}^{(l',k',k),\Phi,L/U} + \mathbf{\Lambda}_{j,:,-}^{(l,i,l',k'),L/U}\mathbf{\Omega}_{j,:}^{(l',k',k),\Phi,U/L})(1 \leq k \leq t),$$

$$\mathbf{\Lambda}_{j,:}^{(l,i,0,k),L/U} = 0 (\forall k \notin P),$$

$$\mathbf{\Delta}_j^{(l,i,0),L/U} = \mathbf{\Delta}_j^{(l,i,l',L/U)} + \sum\nolimits_{k=1}^{n} \mathbf{\Lambda}_{j,:,+}^{(l,i,l',k),L/U}\mathbf{\Theta}_j^{(l',k),\Phi,L/U} + \mathbf{\Lambda}_{j,:,-}^{(l,i,l',k),L/U}\mathbf{\Theta}_j^{(l',k),\Phi,U/L}.$$

We take global bounds as Eq. (3) and Eq. (4) to obtain the bounds of the l-th layer.

Advantageous of Combining the Backward Process with a Forward Process Introducing a forward process can significantly reduce the complexity of verifying Transformers. With the backward process only, we need to compute $\mathbf{\Lambda}^{(l,i,l',k)}$ and $\mathbf{\Delta}^{(l,i,l')}$ $(l' \leq l)$, where the major cost is on $\mathbf{\Lambda}^{(l,i,l',k)}$ and there are $O(m^2n^2)$ such matrices to compute. The $O(n^2)$ factor is from the dependency between all pairs of positions in the input and output respectively, which makes the algorithm inefficient especially when the input sequence is long. In contrast, the forward process represents the bounds as linear functions of the perturbed positions only instead of all positions by computing $\mathbf{\Omega}^{(l,i)}$ and $\mathbf{\Theta}^{(l,i)}$. Imperceptible adversarial examples may not have many perturbed positions (Gao et al., 2018; Ko et al., 2019), and thus we may assume that the number of perturbed positions, t, is small. The major cost is on $\mathbf{\Omega}^{(l,i)}$ while there are only $O(mn)$ such matrices and the sizes of $\mathbf{\Lambda}^{(l,i,l',k)}$ and $\mathbf{\Omega}^{(l,i)}$ are relatively comparable for a small t. We combine the backward process and the forward process. The number of matrices $\mathbf{\Omega}$ is $O(mn)$ in the forward process, and for the backward process, since we do not propagate bounds over self-attention layers and there is no cross-position dependency in other sub-layers, we only compute $\mathbf{\Lambda}^{(l,i,l',k)}$ such that $i = k$, and thus the number of matrices $\mathbf{\Lambda}$ is reduced to $O(m^2n)$. So the total number of matrices $\mathbf{\Lambda}$ and $\mathbf{\Omega}$ we compute is $O(m^2n)$ and is $O(n)$ times smaller than $O(m^2n^2)$ when only the backward process is used. Moreover, the backward process makes bounds tighter compared to solely the forward one, as we show in Appendix D.

4 Experiments

4.1 Datasets and Models

We conduct experiments on two sentiment analysis datasets: Yelp (Zhang et al., 2015) and SST-2 (Socher et al., 2013). Yelp consists of 560,000/38,000 examples in the training/test set and SST-2 consists of 67,349/872/1,821 examples in the training/development/test set. Each example is a sentence or a sentence segment (for the training data of SST-2 only) labeled with a sentiment polarity.

We verify the robustness of Transformers trained from scratch. For the main experiments, we consider N-layer models ($N \leq 3$), with 4 attention heads, hidden sizes of 256 and 512 for self-attention and feed-forward layers respectively, and we use ReLU activations for feed-forward layers. We remove the variance related terms in layer normalization, making Transformers verification bounds

tighter while the clean accuracies remain comparable (see Appendix E for discussions). Although our method can be in principal applied to Transformers with any number of layers, we do not use large-scale pre-trained models such as BERT because they are too challenging to be tightly verified for now. Current state-of-the-art verification methods for feed-forward networks either produce loose bounds for large networks (Zhang et al., 2018; Singh et al., 2019), or do not scale due to computational limits (Raghunathan et al., 2018; Dvijotham et al., 2019); Transformers contain more nonlinear operations than feed-forward networks so they are even more challenging for verification.

4.2 Certified Bounds

Dataset	N	Acc.	ℓ_p	Upper		Lower (IBP)		Lower (Ours)		Ours vs Upper	
				Min	Avg	Min	Avg	Min	Avg	Min	Avg
Yelp	1	91.46	ℓ_1	9.332	13.577	1.1E-4	3.1E-4	0.736	0.925	8%	7%
			ℓ_2	0.708	0.988	1.1E-4	3.1E-4	0.286	0.362	40%	37%
			ℓ_∞	0.119	0.159	1.1E-4	3.1E-4	0.031	0.040	26%	25%
	2	91.54	ℓ_1	11.095	16.522	1.2E-7	2.2E-7	0.331	0.490	3%	3%
			ℓ_2	0.840	1.175	1.2E-7	2.2E-7	0.105	0.144	13%	12%
			ℓ_∞	0.133	0.188	1.2E-7	2.2E-7	0.010	0.013	7%	7%
	3	91.58	ℓ_1	10.745	14.706	5E-10	8E-10	0.231	0.376	2%	3%
			ℓ_2	0.806	1.069	5E-10	8E-10	0.060	0.092	7%	9%
			ℓ_∞	0.132	0.187	5E-10	8E-10	0.005	0.008	4%	4%
SST	1	84.13	ℓ_1	6.887	8.859	2.6E-4	3.0E-4	2.541	2.747	37%	31%
			ℓ_2	0.529	0.655	2.6E-4	3.0E-4	0.408	0.448	77%	68%
			ℓ_∞	0.088	0.110	2.6E-4	3.0E-4	0.032	0.035	36%	32%
	2	83.34	ℓ_1	7.003	8.920	4.5E-7	4.7E-7	1.661	1.711	24%	19%
			ℓ_2	0.536	0.665	4.5E-7	4.7E-7	0.283	0.293	53%	44%
			ℓ_∞	0.089	0.114	4.5E-7	4.7E-7	0.022	0.023	25%	20%
	3	83.29	ℓ_1	6.899	8.484	7E-10	9E-10	1.157	1.186	17%	14%
			ℓ_2	0.527	0.634	7E-10	9E-10	0.190	0.195	36%	31%
			ℓ_∞	0.089	0.108	7E-10	9E-10	0.015	0.015	17%	14%

Table 1: Clean accuracies and computed bounds for 1-position perturbation. Bounds include upper bounds (obtained by an enumeration based method), certified lower bounds by IBP and our method respectively. We also report the gap between upper bounds and our lower bounds (represented as the percentage of lower bounds relative to upper bounds). We compute bounds for each possible option of perturbed positions and report the minimum ("Min") and average ("Avg") among them.

N	Yelp				SST			
	Lower (IBP)		Lower (Ours)		Lower (IBP)		Lower (Ours)	
	Min	Avg	Min	Avg	Min	Avg	Min	Avg
1	5.3E-5	1.1E-4	0.179	0.223	1.1E-4	1.2E-4	0.214	0.233
2	5.8E-8	9.1E-8	0.058	0.079	1.8E-7	1.9E-7	0.138	0.143
3	3.1E-10	5.0E-10	0.032	0.048	3.7E-10	5.8E-10	0.088	0.091

Table 2: Bounds by IBP and our method for 2-position perturbation constrained by ℓ_2-norm.

We compute certified lower bounds for different models on different datasets. We include 1-position perturbation constrained by $\ell_1/\ell_2/\ell_\infty$-norms and 2-position perturbation constrained by ℓ_2-norm. We compare our lower bounds with those computed by the Interval Bound Propagation (IBP) (Gowal et al., 2018) baseline. For 1-position perturbation, we also compare with *upper bounds* computed by enumerating all the words in the vocabulary and finding the word closest to the original one such that the word substitution alters the predicted label. This method has an exponential complexity with respect to the vocabulary size and can hardly be extended to perturbations on 2 or more positions; thus we do not include upper bounds for 2-position perturbation. For each example, we enumerate possible options of perturbed positions (there are $\binom{n}{t}$ options), and we integrate results from different options by taking the minimum or average respectively. We report the average results on 10 correctly classified random test examples with sentence lengths no more than 32 for 1-position perturbation and 16 for 2-position perturbation. Table 1 and Table 2 present the results for 1-position and 2-position perturbation respectively. Our certified lower bounds are significantly larger and thus tighter than those by IBP. For 1-position perturbation, the lower bounds are consistently smaller than the

upper bounds, and the gap between the upper bounds and our lower bounds are reasonable compared with that in previous work on verification of feed-forward networks, e.g. in (Weng et al., 2018; Zhang et al., 2018) the upper bounds are in the order of 10 times larger than lower bounds. This demonstrates that our proposed method can compute robustness bounds for Transformers in a similar quality to the bounds of simpler neural networks.

4.3 Effectiveness of Combining the Backward Process with a Forward Process

Dataset	Acc.	ℓ_p	Fully-Forward			Fully-Backward			Backward & Forward		
			Min	Avg	Time	Min	Avg	Time	Min	Avg	Time
Yelp	91.29	ℓ_1	0.541	1.216	18.5	0.893	2.085	507.1	0.892	2.081	29.7
		ℓ_2	1.060	2.193	16.3	1.599	3.438	505.0	1.597	3.432	35.4
		ℓ_∞	0.570	0.976	16.7	0.884	1.585	471.8	0.883	1.582	30.5
SST-2	81.87	ℓ_1	0.619	1.020	34.4	0.879	1.442	399.5	0.879	1.442	53.7
		ℓ_2	0.902	1.406	28.0	1.278	1.977	396.2	1.278	1.977	60.7
		ℓ_∞	0.408	0.603	25.0	0.600	0.879	392.5	0.600	0.879	53.8

Table 3: Comparison of certified lower bounds and computation time (sec) by different methods.

In the following, we show the effectiveness of combining the backward process with a forward process. We compare our proposed method (*Backward & Forward*) to two variations: *Fully-Forward* propagates bounds in a forward manner for all sub-layers besides self-attention layers; *Fully-Backward* computes bounds for all sub-layers including self-attention layers using the backward bound propagation and without the forward process. We compare the tightness of bounds and computation time of the three methods. We use smaller models with the hidden sizes reduced by half, and we use 1-position perturbation only, to accommodate *Fully-Backward* with large computational cost. Experiments are conducted on an NVIDIA TITAN X GPU. Table 3 presents the results. Bounds by *Fully-Forward* are significantly looser while those by *Fully-Backward* and *Backward & Forward* are comparable. Meanwhile, the computation time of *Backward & Forward* is significantly shorter than that of *Fully-Backward*. This demonstrates that our method of combining the backward and forward process can compute comparably tight bounds much more efficiently.

4.4 Identifying Words Important to Prediction

Method	Importance Score (on SST)	Words Identified from 10 Examples on the Yelp Dataset (split by "/")
		Most Important Words or Symbols
Grad	0.39	**terrible**/**great**/diner/**best**/**best**/food/**slow**/**great**/**perfect**/**best**
Upper	0.48	**terrible**/./.**best**/'/and/**slow**/**great**/this/**best**
Ours	**0.57**	**terrible**/**great**/level/**best**/**best**/**good**/**slow**/**great**/**perfect**/**best**
		Least Important Words or Symbols
Grad	0.46	./**decadent**/././././the/././.
Upper	0.12	eat/diner/**typical**/boba/torta/bit/**new**/,/&/boba
Ours	**0.05**	./our/./././a/the/././.

Table 4: Average importance scores of the most/least important words identified from 100 examples respectively on SST by different methods. For the *most important* words identified, larger important scores are better, and vice versa. Additionally, we show most/least important words identified from 10 examples on the Yelp dataset. Boldfaced words are considered to have strong sentiment polarities, and they should appear as most important words rather than least important ones.

The certified lower bounds can reflect how sensitive a model is to the perturbation of each input token. Intuitively, if a word is more important to the prediction, the model is more sensitive to its perturbation. Therefore, the certified lower bounds can be used to identify important words. In the following, we conduct an experiment to verify whether important words can be identified by our certified lower bounds. We use a 1-layer Transformer classifier under 1-position perturbation constrained by ℓ_2-norm. We compare our method with two baselines that also estimate local vulnerability: *Upper* uses upper bounds; *Gradient* identifies the word whose embedding has the largest ℓ_2-norm of gradients as the most important and vice versa.

Quantitative Analysis on SST SST contains sentiment labels for all phrases on parse trees, where the labels range from very negative (0) to very positive (4), and 2 for neutral. For each word, assuming its label is x, we take $|x - 2|$, i.e. the distance to the neutral label, as the *importance score*, since less neutral words tend to be more important for the sentiment polarity of the sentence. We evaluate on 100 random test input sentences and compute the average importance scores of the most or least important words identified from the examples. In Table 4, compared to the baselines ("Upper" and "Grad"), the average importance score of the most important words identified by our lower bounds are the largest, while the least important words identified by our method have the smallest average score. This demonstrates that our method identifies the most and least important words more accurately compared to baseline methods.

Qualitative Analysis on Yelp We further analyze the results on a larger dataset, Yelp. Since Yelp does not provide per-word sentiment labels, importance scores cannot be computed as on SST. Thus, we demonstrate a qualitative analysis. We use 10 random test examples and collect the words identified as the most and least important word in each example. In Table 4, most words identified as the most important by certified lower bounds are exactly the words reflecting sentiment polarities (bold-faced words), while those identified as the least important words are mostly stopwords. Baseline methods mistakenly identify more words containing no sentiment polarity as the most important. This again demonstrates that our certified lower bounds identify word importance better than baselines and our bounds provide meaningful interpretations in practice. While gradients evaluate the sensitivity of each input word, this evaluation only holds true within a very small neighborhood (where the classifier can be approximated by a first-order Taylor expansion) around the input sentence. Our certified method gives valid lower bounds that hold true within a large neighborhood specified by a perturbation set S, and thus it provides more accurate results.

5 Conclusion

We propose the first robustness verification method for Transformers, and tackle key challenges in verifying Transformers, including cross-nonlinearity and cross-position dependency, for efficient and effective verification. Our method computes certified lower bounds that are significantly tighter than those by IBP. Quantitative and qualitative analyses further show that our bounds are meaningful and can reflect the importance of different words in sentiment analysis.

References

Michael E Akintunde, Andreea Kevorchian, Alessio Lomuscio, and Edoardo Pirovano. Verification of rnn-based neural agent-environment systems. In *Proceedings of the 33th AAAI Conference on Artificial Intelligence (AAAI19). Honolulu, HI, USA. AAAI Press.*, 2019.

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *EMNLP*, pp. 2890–2896, 2018.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pp. 4790–4799, 2018.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pp. 121–138. Springer, 2018.

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

Krishnamurthy Dj Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. *UAI*, 2019.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 31–36, 2018.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 269–286. Springer, 2017.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 50–56. IEEE, 2018.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18. IEEE, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2266–2276, 2017.

Yu-Lun Hsieh, Minhao Cheng, Da-Cheng Juan, Wei Wei, Wen-Lian Hsu, and Cho-Jui Hsieh. On the robustness of self-attentive models. In *ACL*, pp. 1520–1529, 2019.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492*, 2019.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. *arXiv preprint arXiv:1909.00986*, 2019.

Kyle D Julian, Shivam Sharma, Jean-Baptiste Jeannin, and Mykel J Kochenderfer. Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. *arXiv preprint arXiv:1903.00762*, 2019.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 197–206. IEEE, 2018.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. Popqorn: Quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning*, pp. 3468–3477, 2019.

Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wenbing Huang, and Junzhou Huang. Semi-supervised graph classification: A hierarchical graph perspective. In *The World Wide Web Conference*, pp. 972–982. ACM, 2019a.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019b.

Xuankang Lin, He Zhu, Roopsha Samanta, and Suresh Jagannathan. Art: Abstraction refinement-guided training for provably correct neural networks. *arXiv preprint arXiv:1907.10662*, 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3575–3583, 2018.

Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pp. 49–54. IEEE, 2016.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Chongli Qin, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, Pushmeet Kohli, et al. Verification of non-linear specifications for neural networks. *arXiv preprint arXiv:1902.09592*, 2019.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pp. 10877–10887, 2018.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robust verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019.

Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *IJCAI*, 2018.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10825–10836, 2018.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. Vl-bert: Pre-training of generic visual-linguistic representations. *arXiv preprint arXiv:1908.08530*, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Qinglong Wang, Kaixuan Zhang, Xue Liu, and C Lee Giles. Verification of recurrent neural networks through rule extraction. *arXiv preprint arXiv:1811.06029*, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pp. 6367–6377, 2018b.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018c.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5283–5292, 2018.

Yichen Yang and Martin Rinard. Correctness verification of neural networks. *arXiv preprint arXiv:1906.01030*, 2019.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

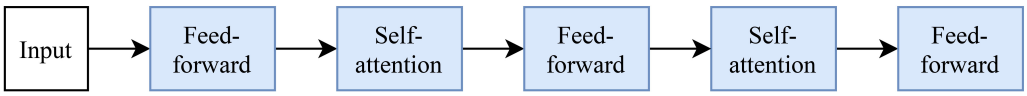
Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*, pp. 4939–4948, 2018.

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5757–5764, 2019.

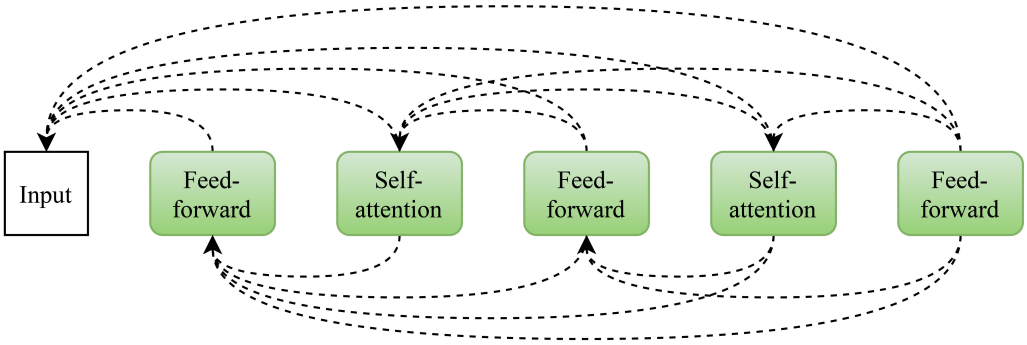
Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342*, 2017.

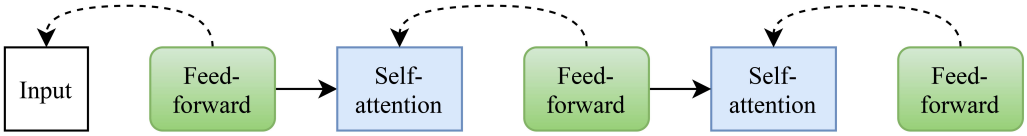
A ILLUSTRATION OF DIFFERENT BOUNDING PROCESSES



(a) Fully-Forward



(b) Fully-Backward



(c) Backward & Forward

Figure 1: Illustration of three different bounding processes: Fully-Forward (a), Fully-Backward (b), and Backward&Forward (c). We show an example of a 2-layer Transformer, where operations can be divided into two kinds of blocks, "Feed-forward" and "Self-attention". "Self-attention" contains operations in the self-attention mechanism starting from queries, keys, and values, and "Feed-forward" contains all the other operations including linear transformations and unary nonlinear functions. Arrows with solid lines indicate the propagation of linear bounds in a forward manner. Each backward arrow $A_k \rightarrow B_k$ with a dashed line for blocks A_k, B_k indicates that there is a backward bound propagation to block B_k when computing bounds for block A_k. Blocks with blue rectangles have forward processes inside the blocks, while those with green rounded rectangles have backward processes inside.

Figure 1 illustrates a comparison of the Fully-Forward, Fully-Backward and Backward & Forward processes, for a 2-layer Transformer as an example. For Fully-Forward, there are only forward processes connecting adjacent layers and blocks. For Fully-Backward, there are only backward processes, and each layer needs a backward bound propagation to all the previous layers. For our Backward & Forward algorithm, we use backward processes for the feed-forward parts and forward processes for self-attention layers, and for layers after self-attention layers, they no longer need backward bound propagation to layers prior to self-attention layers. In this way, we resolve the cross-position dependency in verifying Transformers while still keeping bounds comparably tight as those by using fully backward processes. Empirical comparison of the three frameworks are presented in Sec. 4.3.

B LINEAR BOUNDS OF UNARY NONLINEAR FUNCTIONS

We show in Sec. 3.2 that linear bounds can be propagated over unary nonlinear functions as long as the unary nonlinear functions can be bounded with linear functions. Such bounds are determined for each neuron respectively, according to the bounds of the input for the function. Specifically, for a unary nonlinear function $\sigma(x)$, with the bounds of x obtained previously as $x \in [l, u]$, we aim to derive a linear lower bound $\alpha^L x + \beta^L$ and a linear upper bound $\alpha^U x + \beta^U$, such that

$$\alpha^L x + \beta^L \leq \sigma(x) \leq \alpha^U x + \beta^U \quad (\forall x \in [l, u])$$

where parameters $\alpha^L, \beta^L, \alpha^U, \beta^U$ are dependent on l, u and designed for different functions $\sigma(x)$ respectively. We introduce how the parameters are determined for different unary nonlinear functions involved in Transformers such that the linear bounds are valid and as tight as possible. Bounds of ReLU and tanh has been discussed by Zhang et al. (2018), and we further derive bounds of e^x, $\frac{1}{x}$, x^2, $\sqrt{x}$. x^2 and $\sqrt{x}$ is only used when the layer normalization is not modified for experiments to study the impact of our modification. For the following description, we define the endpoints of the function to be bounded within the range as $(l, \sigma(l))$ and $(u, \sigma(u))$. We describe how the lines corresponding to the linear bounds of different functions can be determined, and thereby parameters $\alpha^L, \beta^L, \alpha^U, \beta^U$ can be determined accordingly.

ReLU For ReLU activation, $\sigma(x) = \max(x, 0)$. ReLU $\sigma(x)$ is inherently linear on segments $(-\infty, 0]$ and $[0, \infty)$ respectively, so we make the linear bounds exactly $\sigma(x)$ for $u \leq 0$ or $l \geq 0$; and for $l < 0 < u$, we take the line passing the two endpoints as the upper bound; and take $\sigma^L(x) = 0$ when $u < |l|$ and $\sigma^L(x) = 1$ when $u \geq |l|$ as the lower bound, to minimize the gap between the lower bound and the original function.

Tanh For $\tanh$ activation, $\sigma(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$. $\tanh$ is concave for $l \geq 0$, and thus we take the line passing the two endpoints as the lower bound and take a tangent line passing $((l+u)/2, \sigma((l+u)/2)$ as the upper bound. For $u \leq 0$, $\tanh$ is convex, and thus we take the line passing the two endpoints as the upper bound and take a tangent line passing $((l+u)/2, \sigma((l+u)/2)$ as the lower bound. For $l < 0 < u$, we take a tangent line passing the right endpoint and $(d^L, \sigma(d^L))(d^L \leq 0)$ as the lower bound, and take a tangent line passing the left endpoint and $(d^U, \sigma(d^U))(d^U \leq 0)$ as the upper bound. d^L and d^U can be found with a binary search.

Exp $\sigma(x) = exp(x) = e^x$ is convex, and thus we take the line passing the two endpoints as the upper bound and take a tangent line passing $(d, \sigma(d))$ as the lower bound. Preferably, we take $d = (l+u)/2$. However, e^x is always positive and used in the softmax for computing normalized attention probabilities in self-attention layers, i.e. $exp(\mathbf{S}^{(l)}_{i,j})$ and $\sum_{k=1}^{n} exp(\mathbf{S}^{(l)}_{i,k})$. $\sum_{k=1}^{n} exp(\mathbf{S}^{(l)}_{i,k})$ appears in the denominator of the softmax, and to make reciprocal function $\frac{1}{x}$ finitely bounded the range of x should not pass 0. Therefore, we impose a constraint to force the lower bound function to be always positive, i.e. $\sigma^L(l) > 0$, since $\sigma^L(l)$ is monotonously increasing. $\sigma^L_d(x) = e^d(x-d)+e^d$ is the tangent line passing $(d, \sigma(d))$. So the constraint $\sigma^L_d(l) > 0$ yields $d < l+1$. Hence we take $d = \min((l+u)/2, l+1-\Delta_d)$ where Δ_d is a small real value to ensure that $d < l+1$ such as $\Delta_d = 10^{-2}$.

Reciprocal For the reciprocal function, $\sigma(x) = \frac{1}{x}$. It is used in the softmax and layer normalization and its input is limited to have $l > 0$ by the lower bounds of $exp(x)$, and $\sqrt{x}$. With $l > 0$, $\sigma(x)$ is convex. Therefore, we take the line passing the two endpoints as the upper bound. And we take the tangent line passing $((l+u)/2, \sigma((l+u)/2))$ as the lower bound.

Square For the square function, $\sigma(x) = x^2$. It is convex and we take the line passing the two endpoints as the upper bound. And we tan a tangent line passing $(d, \sigma(d))(d \in [l, u])$ as the lower bound. We still prefer to take $d = (l+u)/2$. x^2 appears in computing the variance in layer normalization and is later passed to a square root function to compute a standard derivation. To make the input to the square root function valid, i.e. non-negative, we impose a constraint $\sigma^L(x) \geq 0 (\forall x \in [l, u])$. $\sigma^L_d(x) = 2d(x-d) + d^2$ is the tangent line passing $(d, \sigma(d))$. For $u \leq 0$, x^2 is monotonously decreasing, the constraint we impose is equivalent to $\sigma^L(u) = 2du - d^2 \geq 0$, and with $d \leq 0$, we have $d \geq 2u$. So we take $d = \max((l+u)/2, 2u)$. For $l \geq 0$, x^2 is monotonously increasing, the

constraint we impose is equivalent to $\sigma^L(l) = 2dl - d^2 \geq 0$, and with $d \geq 0$, we have $d \leq 2l$. So we take $d = \max((l+u)/2, 2l)$. $\sigma_d^L(0) = -d^2$ is negative for $d \neq 0$ and is zero for $d = 0$. And for $l < 0 < u$, the constraint we impose is equivalent to $\sigma^L(0) = -d^2 \geq 0$, and thus we take $d = 0$.

Square root For the square root function, $\sigma(x) = \sqrt{x}$. It is used the to compute a standard derivation in the layer normalization and its input is limited to be positive by the lower bounds of x^2 and a smoothing constant, so $l > 0$. $\sigma(x)$ is concave, and thus we take the line passing the two endpoints as the lower bound and take the tangent line passing $((l+u)/2, \sigma((l+u)/2))$ as the upper bound.

C Linear Bounds of Multiplications and Divisions

We provide a mathematical proof of optimal parameters for linear bounds of multiplications used in Sec. 3.3. We also show that linear bounds of division can be indirectly obtained from bounds of multiplications and the reciprocal function.

For each multiplication, we aim to bound $z = xy$ with two linear bounding planes $z^L = \alpha^L x + \beta^L y + \gamma^L$ and $z^U = \alpha^U x + \beta^U y + \gamma^U$, where x and y are both variables and $x \in [l_x, u_x], y \in [l_y, u_y]$ are concrete bounds of x, y obtained from previous layers, such that:

$$z^L = \alpha^L x + \beta^L y + \gamma^L \leq z = xy \leq z^U = \alpha^U x + \beta^U y + \gamma^U \quad \forall (x, y) \in [l_x, u_x] \times [l_y, u_y]$$

Our goal is to determine optimal parameters of bounding planes, i.e. $\alpha^L, \beta^L, \gamma^L, \alpha^U, \beta^U, \gamma^U$, such that the bounds are as tight as possible.

C.1 Lower Bound of Multiplications

We define a difference function $F^L(x, y)$ which is the difference between the original function $z = xy$ and the lower bound $z^L = \alpha^L x + \beta^L y + \gamma^L$:

$$F^L(x, y) = xy - (\alpha^L x + \beta^L y + \gamma^L)$$

To make the bound as tight as possible, we aim to minimize the integral of the difference function $F^L(x, y)$ on our *concerned area* $(x, y) \in [l_x, u_x] \times [l_y, u_y]$, which is equivalent to maximizing

$$V^L = \int_{x \in [l_x, u_x]} \int_{y \in [l_y, u_y]} \alpha^L x + \beta^L y + \gamma^L \tag{7}$$

while $F^L(x, y) \geq 0$ $(\forall (x, y) \in [l_x, u_x] \times [l_y, u_y])$. For an optimal bounding plane, there must exist a point $(x_0, y_0) \in [l_x, u_x] \times [l_y, u_y]$ such that $F^L(x_0, y_0) = 0$ (otherwise we can validly increase γ^L to make V^L larger). To ensure that $F^L(x, y) \geq 0$ within the concerned area, we need to ensure that the minimum value of $F^L(x, y)$ is be non-negative. We show that we only need to check cases when (x, y) is any of $(l_x, l_y), (l_x, u_y), (u_x, l_y), (u_x, u_y)$, i.e. points at the corner of the considered area. The partial derivatives of F^L are:

$$\frac{\partial F^L}{\partial x} = y - \alpha^L$$

$$\frac{\partial F^L}{\partial y} = x - \beta^L$$

If there is $(x_1, y_1) \in (l_x, u_x) \times (l_y, u_y)$ such that $F^L(x_1, y_1) \leq F(x, y)$ $(\forall (x, y) \in [l_x, u_x] \times [l_y, u_y])$, $\frac{\partial F^L}{\partial x}(x_1, y_1) = \frac{\partial F^L}{\partial y}(x_1, y_1) = 0$ should hold. Thereby $\frac{\partial F^L}{\partial x}(x, y), \frac{\partial F^L}{\partial y}(x, y) < 0$ $(\forall (x, y) \in [l_x, x_1) \times [l_y, y_1))$, and thus $F^L(l_x, l_y) < F^L(x_1, y_1)$ and (x_1, y_1) cannot be the point with the minimum value of $F^L(x, y)$. On the other hand, if there is $(x_1, y_1)(x_1 = l_x, y_1 \in (l_y, u_y))$, i.e. on one border of the concerned area but not on any corner, $\frac{\partial F^L}{\partial y}(x_1, y_1) = 0$ should hold. Thereby $\frac{\partial F^L}{\partial y}(x, y) = \frac{\partial F^L}{\partial y}(x_1, y) = 0$ $(\forall (x, y), x = x_1 = l_x)$. So $F^L(x_1, y_1) = F^L(x_1, l_y) = F^L(l_x, l_y)$. This property holds for the other three borders of the concerned area. Therefore, other

points within the concerned area cannot have smaller function value $F^L(x, y)$, so we only need to check the corners, and the constraints on $F^L(x, y)$ become

$$\begin{cases} F^L(x_0, y_0) & = 0 \\ F^L(l_x, l_y) & \geq 0 \\ F^L(l_x, u_y) & \geq 0 \\ F^L(u_x, l_y) & \geq 0 \\ F^L(u_x, u_y) & \geq 0 \end{cases}$$

equivalent to

$$\begin{cases} \gamma^L = x_0 y_0 - \alpha^L x_0 - \beta^L y_0 \\ l_x l_y - \alpha^L (l_x - x_0) - \beta^L (l_y - y_0) - x_0 y_0 \geq 0 \\ l_x u_y - \alpha^L (l_x - x_0) - \beta^L (u_y - y_0) - x_0 y_0 \geq 0 \\ u_x l_y - \alpha^L (u_x - x_0) - \beta^L (l_y - y_0) - x_0 y_0 \geq 0 \\ u_x u_y - \alpha^L (u_x - x_0) - \beta^L (u_y - y_0) - x_0 y_0 \geq 0 \end{cases} \tag{8}$$

We substitute γ^L in Eq. (7) with Eq. (8), yielding

$$V^L = V_0[(l_x + u_x - 2x_0)\alpha^L + (l_y + u_y - 2y_0)\beta^L + 2x_0 y_0]$$

where $V_0 = \frac{(u_x - l_x)(u_y - l_y)}{2}$.

We have shown that the minimum function value $F^L(x, y)$ within the concerned area cannot appear in $(l_x, u_x) \times (l_y, u_y)$, i.e. it can only appear at the border, while (x_0, y_0) is a point with a minimum function value $F^L(x_0, y_0) = 0$, (x_0, y_0) can also only be chosen from the border of the concerned area. At least one of $x_0 = l_x$ and $x_0 = u_x$ holds.

If we take $x_0 = l_x$:

$$V_1^L = V_0[(u_x - l_x)\alpha^L + (l_y + u_y - 2y_0)\beta^L + 2l_x y_0]$$

And from Eq. (8) we obtain

$$\alpha^L \leq \frac{u_x l_y - l_x y_0 - \beta^L (l_y - y_0)}{u_x - l_x}$$

$$\alpha^L \leq \frac{u_x u_y - l_x y_0 - \beta^L (u_y - y_0)}{u_x - l_x}$$

$$l_x \leq \beta^L \leq l_x \Leftrightarrow \beta^L = l_x$$

Then

$$V_1^L = V_0[(u_x - l_x)\alpha^L + l_x(l_y + u_y)]$$

$$\begin{aligned} (u_x - l_x)\alpha^L &\leq -l_x y_0 + \min(u_x l_y - \beta^L (l_y - y_0), u_x u_y - \beta^L (u_y - y_0)) \\ &= -l_x y_0 + \min(u_x l_y - l_x (l_y - y_0), u_x u_y - l_x (u_y - y_0)) \\ &= (u_x - l_x) \min(l_y, u_y) \\ &= (u_x - l_x) l_y \end{aligned}$$

So

$$\alpha^L \leq l_y$$

To maximize V_1^L, since now only α^L is unknown in V_1^L and the coefficient of α^L, $V_0(u_x - l_x) \geq 0$, we take $\alpha^L = l_y$, and then

$$V_1^L = V_0(u_x l_y + l_x u_y)$$

is a constant.

For the other case if we take $x_0 = u_x$:

$$V_2^L = V_0[(l_x - u_x)\alpha^L + (l_y + u_y - 2y_0)\beta^L + 2u_x y_0]$$

$$\alpha^L \geq \frac{l_x l_y - u_x y_0 - \beta^L (l_y - y_0)}{l_x - u_x}$$

$$\alpha^L \geq \frac{l_x u_y - u_x y_0 - \beta^L (u_y - y_0)}{l_x - u_x}$$
$$u_x \leq \beta^L \leq u_x \Leftrightarrow \beta^L = u_x$$
$$V_2^L = V_0[(l_x - u_x)\alpha^L + u_x(l_y + u_y)]$$

$$\begin{aligned}(l_x - u_x)\alpha^L &\leq -u_x y_0 + \min(l_x l_y - \beta^L(l_y - y_0), l_x u_y - \beta^L(u_y - y_0)) \\ &= \min(l_x l_y - u_x l_y, l_x u_y - u_x u_y) \\ &= (l_x - u_x)\max(l_y, u_y) \\ &= (l_x - u_x)u_y\end{aligned}$$

So

$$\alpha^L \geq u_y$$

We take $\alpha^L = u_y$ similarly as in the case when $x_0 = l_x$, and then

$$V_2^L = V_0(l_x u_y + u_x l_y)$$

We notice that $V_1^L = V_2^L$, so we can simply adopt the first one. We also notice that V_1^L, V_2^L are independent of y_0, so we may take any y_0 within $[l_y, u_y]$ such as $y_0 = l_y$. Thereby, we obtain the a group of optimal parameters of the lower bounding plane:

$$\begin{cases} \alpha^L &= l_y \\ \beta^L &= l_x \\ \gamma^L &= -l_x l_y \end{cases}$$

C.2 Upper Bound of Multiplications

We derive the upper bound similarly. We aim to minimize

$$V^U = V_0[(l_x + u_x - 2x_0)\alpha^U + (l_y + u_y - 2y_0)\beta^U + 2x_0 y_0]$$

where $V_0 = \frac{(u_x - l_x)(u_y - l_y)}{2}$.

If we take $x_0 = l_x$:

$$V_1^U = V_0[(u_x - l_x)\alpha^U + (l_y + u_y - 2y_0)\beta^U + 2l_x y_0]$$

$$\alpha^U \geq \frac{u_x l_y - l_x y_0 - \beta^U (l_y - y_0)}{u_x - l_x}$$
$$\alpha^U \geq \frac{u_x u_y - l_x y_0 - \beta^U (u_y - y_0)}{u_x - l_x}$$
$$l_x \leq \beta^U \leq l_x \Leftrightarrow \beta^U = l_x$$

Then

$$V_1^U = V_0[(u_x - l_x)\alpha^U + l_x(l_y + u_y)]$$

$$\begin{aligned}(u_x - l_x)\alpha^U &\geq -l_x y_0 + \max(u_x l_y - \beta^U(l_y - y_0), u_x u_y - \beta^U(u_y - y_0)) \\ &= \max(u_x l_y - l_x l_y, u_x u_y - l_x u_y) \\ &= (u_x - l_x)\max(l_y, u_y) \\ &= (u_x - l_x)u_y\end{aligned}$$

So

$$\alpha^U \geq u_y$$

To minimize V_1^U, we take $\alpha^U = u_y$, and then

$$V_1^U = V_0(l_x l_y + u_x u_y)$$

For the other case if we take $x_0 = u_x$:

$$V_2^U = V_0[(l_x - u_x)\alpha^U + (l_y + u_y - 2y_0)\beta^U + 2u_x y_0]$$
$$\alpha^U \leq \frac{l_x l_y - u_x y_0 - \beta^U(l_y - y_0)}{l_x - u_x}$$
$$\alpha^U \leq \frac{l_x u_y - u_x y_0 - \beta^U(u_y - y_0)}{l_x - u_x}$$
$$u_x \leq \beta^U \leq u_x \Leftrightarrow \beta^U = u_x$$

Then

$$V_2^U = V_0[(l_x - u_x)\alpha^U + u_x(l_y + u_y)]$$

$$\begin{aligned}(l_x - u_x)\alpha^U &\geq -u_x y_0 + \max(l_x l_y - \beta^U(l_y - y_0), l_x u_y - \beta^U(u_y - y_0)) \\ &= \max(l_x l_y - u_x l_y, l_x u_y - u_x u_y) \\ &= (l_x - u_x)\min(l_y, u_y) \\ &= (l_x - u_x) l_y\end{aligned}$$

So

$$\alpha^U \leq l_y$$

To minimize V_2^U, we take $\alpha^U = l_y$, and then

$$V_2^L = V_0(l_x l_y + u_x u_y)$$

Since $V_1^U = V_2^U$, we simply adopt the first case. And V_1^U, V_2^U are independent of y_0, so we may take any y_0 within $[l_y, u_y]$ such as $y_0 = l_y$. Thereby, we obtain a group of optimal parameters of the upper bounding plane:

$$\begin{cases} \alpha^U & = u_y \\ \beta^U & = l_x \\ \gamma^U & = -l_x u_y \end{cases}$$

C.3 Linear Bounds of Divisions

We have shown that closed-form linear bounds of multiplications can be derived. However, we find that directly bounding $z = \frac{x}{y}$ is tricky. If we try to derive a lower bound $z^L = \alpha^L x \beta^L y + \gamma$ for $z = \frac{x}{y}$ as Appendix C.1, the difference function is

$$F^L(x, y) = \frac{x}{y} - (\alpha^L x + \beta^L y + \gamma^L)$$

The partial derivatives of F^L are:

$$\frac{\partial F^L}{\partial x} = \frac{1}{y} - \alpha^L$$
$$\frac{\partial F^L}{\partial y} = -\frac{x}{y^2} - \beta^L$$

If there is $(x_1, y_1) \in (l_x, u_x) \times (l_y, u_y)$ such that $\frac{\partial F^L}{\partial x}(x, y) = \frac{\partial F^L}{\partial y}(x, y) = 0$, unlike that the case for multiplications, none of areas $[l_x, x_1) \times [l_y, y_1)$, $[l_x, x_1) \times (y_1, u_y]$, $(x_1, u_x] \times [l_y, y_1)$ and $(x_1, u_x] \times (y_1, u_y]$ has guarantees that $\frac{\partial F^L}{\partial x}(x, y) = \frac{\partial F^L}{\partial y}(x, y) < 0$ for all (x, y) with in the corresponding area. It is possible that (x_1, y_1) does have a minimum function value $F^L(x_1, y_1)$ among all $(x, y) \in [l_x, u_x] \times [l_y, u_y]$, i.e. a minimum function value of $F^L(x, y)$ for (x, y) within the concerned area may appear at a point other than the corners. For example, for $l_x = 0.05, u_x = 0.15, l_y = 0.05, u_y = 0.15$, $\alpha = 10, \beta = -20, \gamma = 2$, the minimum function value of $F^L(x, y)$ for $(x, y) \in [0.05, 0.15] \times [0.05, 0.15]$ appears at $(0.1, 0.1)$ which is not a corner of $[0.05, 0.15] \times [0.05, 0.15]$. This makes it more difficult to derive closed-form parameters such that the constraints on $F^L(x, y)$ are satisfied. Fortunately, we can bound $z = \frac{x}{y}$ indirectly by utilizing bounds of multiplications and reciprocal functions. We bound $z = \frac{x}{y}$ by first bounding a unary function $\overline{y} = \frac{1}{y}$ and then bounding the multiplication $z = x\overline{y}$.

D TIGHTNESS OF BOUNDS BY THE BACKWARD PROCESS AND FORWARD PROCESS

We have discussed that combining the backward process with a forward process can reduce computational complexity, compared to the method with the backward process only. But we only use the forward process for self-attention layers and do not fully use the forward process for all sublayers, because bounds by the forward process can be looser than those by the backward process. We compare the tightness of bounds by the forward process and the backward process respectively. To illustrate the difference, for simplicity, we consider a m-layer feed-forward network $\Phi^{(0)} = \mathbf{x},\ \mathbf{y}^{(l)} = \mathbf{W}^{(l)}\Phi^{(l-1)}(\mathbf{x}) + \mathbf{b}^{(l)}, \Phi^{(l)}(\mathbf{x}) = \sigma(\mathbf{y}^{(l)}(\mathbf{x}))(0 < l \leq m)$, where $\mathbf{x}$ is the input vector, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and the bias vector for the l-th layer, $\mathbf{y}^{(l)}(\mathbf{x})$ is the pre-activation vector of the l-th layer, $\Phi^{(l)}(\mathbf{x})$ is the vector of neurons in the l-th layer, and $\sigma(\cdot)$ is an activation function. Before taking global bounds, both the backward process and the forward process bound $\Phi_j^{(l)}(\mathbf{x})$ with linear functions of $\mathbf{x}$. When taking global bounds as Eq. (3) and Eq. (4), only the norm of weight matrix is directly related to the ϵ in the binary search for certified lower bounds. Therefore, we measure the tightness of the computed bounds using the difference between weight matrices for lower bounds and upper bounds respectively. We show how it is computed for the forward process and the backward process respectively.

D.1 THE FORWARD PROCESS

For the forward process, we bound each neuron $\Phi_j^{(l)}(\mathbf{x})$ with linear functions:

$$\mathbf{\Omega}_{j,:}^{(l),L}\mathbf{x} + \mathbf{\Theta}_j^{(l),L} \leq \Phi_j^{(l)}(\mathbf{x}) \leq \mathbf{\Omega}_{j,:}^{(l),U}\mathbf{x} + \mathbf{\Theta}_j^{(l),U}$$

To measure the tightness of the bounds, We are interested in $\mathbf{\Omega}^{(l),L}$, $\mathbf{\Omega}^{(l),U}$, and also $\mathbf{\Omega}^{(l),U} - \mathbf{\Omega}^{(l),L}$. Initially,

$$\mathbf{\Omega}^{(0),L/U} = \mathbf{I},\ \ \mathbf{\Theta}^{(0),L/U} = \mathbf{0},\ \ \mathbf{\Omega}^{(0),U} - \mathbf{\Omega}^{(0),L} = \mathbf{0}$$

We can forward propagate the bounds of $\Phi^{(l-1)}(\mathbf{x})$ to $\mathbf{y}^{(l)}(\mathbf{x})$:

$$\mathbf{\Omega}_{j,:}^{(l),y,L}\mathbf{x} + \mathbf{\Theta}_j^{(l),y,L} \leq \mathbf{y}_j^{(l)}(\mathbf{x}) \leq \mathbf{\Omega}_{j,:}^{(l),y,U}\mathbf{x} + \mathbf{\Theta}_j^{(l),y,U}$$

where

$$\mathbf{\Omega}_{j,:}^{(l),y,L/U} = \sum_{\mathbf{W}_{j,i}^{(l)}>0} \mathbf{W}_{j,i}^{(l)}\mathbf{\Omega}_{i,:}^{(l-1),L/U} + \sum_{\mathbf{W}_{j,i}^{(l)}<0} \mathbf{W}_{j,i}^{(l)}\mathbf{\Omega}_{i,:}^{(l-1),U/L}$$

$$\mathbf{\Theta}^{(l),y,L/U} = \sum_{\mathbf{W}_{j,i}^{(l)}>0} \mathbf{W}_{j,i}^{(l)}\mathbf{\Theta}_i^{(l-1),L/U} + \sum_{\mathbf{W}_{j,i}^{(l)}<0} \mathbf{W}_{j,i}^{(l)}\mathbf{\Theta}_i^{(l-1),U/L} + \mathbf{b}^{(l)}$$

With global bounds of $\mathbf{y}^{(l)}(\mathbf{x})$ which can be obtained as Eq. (3) and Eq. (4), we bound the activation function:

$$\alpha_j^{(l),L}\mathbf{y}^{(l)}(\mathbf{x}) + \beta_j^{(l),L} \leq \sigma(\mathbf{y}_j^{(l)}(\mathbf{x})) \leq \alpha_j^{(l),U}\mathbf{y}^{(l)}(\mathbf{x}) + \beta_j^{(l),U}$$

And then bounds can be propagated from $\Phi^{(l-1)}(\mathbf{x})$ to $\Phi^{(l)}(\mathbf{x})$:

$$\mathbf{\Omega}_{j,:}^{(l),L/U} = \begin{cases} \alpha_j^{(l),L/U}\mathbf{\Omega}_{j,:}^{(l),y,L/U} & \alpha_j^{(l),L/U} \geq 0 \\ \alpha_j^{(l),L/U}\mathbf{\Omega}_{j,:}^{(l),y,U/L} & \alpha_j^{(l),L/U} < 0 \end{cases}$$

$$\mathbf{\Theta}_j^{(l),L/U} = \begin{cases} \alpha_j^{(l),L/U}\mathbf{\Theta}_j^{(l),y,L/U} + \beta_j^{(l),L/U} & \alpha_j^{(l),L/U} \geq 0 \\ \alpha_j^{(l),L/U}\mathbf{\Theta}_j^{(l),y,U/L} + \beta_j^{(l),L/U} & \alpha_j^{(l),L/U} < 0 \end{cases}$$

Therefore,

$$\mathbf{\Omega}_{j,:}^{(l),U} - \mathbf{\Omega}_{j,:}^{(l),L} = (\alpha_j^{(l),U} - \alpha_j^{(l),L})|\mathbf{W}_j^{(l)}|(\mathbf{\Omega}_{j,:}^{(l-1),U} - \mathbf{\Omega}_{j,:}^{(l-1),L}) \tag{9}$$

illustrates how the tightness of the bounds is changed from earlier layers to later layers.

D.2 THE BACKWARD PROCESS AND DISCUSSIONS

For the backward process, we bound the neurons in the l-th layer with linear functions of neurons in a previous layer, the l'-th layer:

$$\Phi^{(l,l'),L} = \mathbf{\Lambda}_{j,:}^{(l,l'),L}\Phi^{(l')}(\mathbf{x}) + \mathbf{\Delta}_j^{(l,l'),L} \leq \Phi^{(l)}(\mathbf{x}) \leq \mathbf{\Lambda}_{j,:}^{(l,l'),U}\Phi^{(l')}(\mathbf{x}) + \mathbf{\Delta}_j^{(l,l'),U} = \Phi^{(l,l'),U}$$

We have shown in Sec. 3.1 how such bounds can be propagated to $l' = 0$, for the case when the input is sequential. For the nonsequential case we consider here, it can be regarded as a special case when the input length is 1. So we can adopt the method in Sec. 3.1 to propagate bounds for the feed-forward network we consider here. We are interested in $\mathbf{\Lambda}^{(l,l'),L}$, $\mathbf{\Lambda}^{(l,l'),U}$ and also $\mathbf{\Lambda}^{(l,l'),U} - \mathbf{\Lambda}^{(l,l'),L}$. Weight matrices of linear bounds before taking global bounds are $\mathbf{\Lambda}^{(l,0),L}$, $\mathbf{\Lambda}^{(l,0),U}$ which are obtained by propagating the bounds starting from $\mathbf{\Lambda}^{(l,l),L} = \mathbf{\Lambda}^{(l,l),U} = \mathbf{I}$. According to bound propagation described in Sec. 3.2,

$$\mathbf{\Lambda}_{:,j}^{(l,l'-1),U} - \mathbf{\Lambda}_{:,j}^{(l,l'-1),L} = (\alpha_j^{(l'),U}(\mathbf{\Lambda}_{:,j,+}^{(l,l'),U} - \mathbf{\Lambda}_{:,j,-}^{(l,l'),L}) - \alpha_j^{(l'),L}(\mathbf{\Lambda}_{:,j,+}^{(l,l'),L} - \mathbf{\Lambda}_{:,j,-}^{(l,l'),U}))\mathbf{W}^{(l')} \quad (10)$$

illustrates how the tightness bounds can be measured during the backward bound propagation until $l' = 0$.

There is a $\mathbf{W}^{(l')}$ in Eq. (10) instead of $|\mathbf{W}^{(l')}|$ in Eq. (9). The norm of $(\mathbf{\Omega}_{j,:}^{(l),U} - \mathbf{\Omega}_{j,:}^{(l),L})$ in Eq. (9) can easily grow larger as l increases during the forward propagation when $\|\mathbf{W}_j^{(l)}\|$ is greater than 1, while this generally holds for neural networks to have $\|\mathbf{W}_j^{(l)}\|$ greater than 1 in feed-forward layers. While in Eq. (10), $\mathbf{W}_j^{(l')}$ can have both positive and negative elements and tends to allow cancellations for different $\mathbf{W}_{j,i}^{(l')}$, and thus the norm of $(\mathbf{\Lambda}_{:,j}^{(l,l'-1),U} - \mathbf{\Lambda}_{:,j}^{(l,l'-1),L})$ tends to be smaller. Therefore, the bounds computed by the backward process tend to be tighter than those by the forward framework, which is consistent with our experiment results in Table 3.

E IMPACT OF MODIFYING THE LAYER NORMALIZATION

The original Transformers have a layer normalization after the embedding layer, and two layer normalization before and after the feed-forward part in each Transformer layer. We modify the layer normalization, $f(\mathbf{x}) = \mathbf{w}(\mathbf{x} - \mu)/\sigma + \mathbf{b}$, where $\mathbf{x}$ is d-dimensional a vector to be normalized, μ and σ are the mean and standard derivation of $\{\mathbf{x}_i\}$ respectively, and $\mathbf{w}$ and $\mathbf{b}$ are gain and bias parameters respectively. $\sigma = \sqrt{(1/d)\sum_{i=1}^{d}(\mathbf{x}_i - \mu)^2 + \epsilon_s}$ where ϵ_s is a smoothing constant. It involves $(\mathbf{x}_i - \mu)^2$ whose linear lower bound is loose and exactly 0 when the range of the $\mathbf{x}_i - \mu$ passes 0. When the ℓ_p norm of the perturbation is relatively larger, there can be many $\mathbf{x}_i - \mu$ with ranges passing 0, which can cause the lower bound of σ to be small and thereby the upper bound of $f_i(\mathbf{x})$ to be large. This can make the certified bounds loose. To resolve this, we modify the layer normalization into $f(\mathbf{x}) = \mathbf{w}(\mathbf{x} - \mu) + \mathbf{b}$ by removing the standard derivation term. We use an experiment to study the impact of this modification. We compare the clean accuracies and certified bounds of the models with modified layer normalization to models with standard layer normalization and with no layer normalization respectively. Table 5 presents the results. Certified lower bounds of models with no layer normalization or our modification are significantly tighter than those of corresponding models with the standard layer normalization. Meanwhile, the clean accuracies of models with our modification are comparable with those of the original model, and particularly higher than the 1-layer model with no layer normalization on SST. This demonstrates that it is worthwhile to modify the layer normalization in Transformers, to make the models better to be verified while keeping comparable clean accuracies.

Dataset	N	LayerNorm	Acc.	ℓ_p	Upper		Lower		Ours vs Upper	
					Min	Avg	Min	Avg	Min	Avg
Yelp	1	Standard	91.64	ℓ_1	188.305	197.873	0.012	0.023	6.6E-5	1.2E-4
				ℓ_2	15.115	15.363	0.010	0.020	6.7E-4	1.3E-3
				ℓ_∞	2.029	2.932	0.002	0.005	9.5E-4	1.8E-3
		None	91.46	ℓ_1	8.023	12.479	0.764	0.953	10%	8%
				ℓ_2	0.583	0.878	0.298	0.365	51%	42%
				ℓ_∞	0.084	0.131	0.030	0.038	36%	29%
		Ours	91.46	ℓ_1	9.332	13.577	0.736	0.925	8%	7%
				ℓ_2	0.708	0.988	0.286	0.362	40%	37%
				ℓ_∞	0.119	0.159	0.031	0.040	26%	25%
	2	Standard	91.96	ℓ_1	191.899	201.829	0.003	0.005	1.7E-5	2.3E-5
				ℓ_2	15.281	15.543	0.002	0.003	1.3E-4	2.1E-4
				ℓ_∞	2.002	3.033	0.000	0.001	1.4E-4	1.7E-4
		None	91.59	ℓ_1	8.145	15.021	0.603	0.751	7%	5%
				ℓ_2	0.592	1.034	0.134	0.168	23%	16%
				ℓ_∞	0.088	0.136	0.011	0.014	13%	10%
		Ours	91.54	ℓ_1	11.095	16.522	0.331	0.490	3%	3%
				ℓ_2	0.840	1.175	0.105	0.144	13%	12%
				ℓ_∞	0.133	0.188	0.010	0.013	7%	7%
SST	1	Standard	84.06	ℓ_1	191.424	195.492	0.008	0.015	4.1E-5	7.4E-5
				ℓ_2	15.549	15.632	0.006	0.012	3.8E-4	7.9E-4
				ℓ_∞	2.262	2.503	0.001	0.003	4.6E-4	1.2E-3
		None	82.58	ℓ_1	7.282	8.398	2.680	2.838	37%	34%
				ℓ_2	0.549	0.623	0.437	0.474	80%	76%
				ℓ_∞	0.089	0.106	0.034	0.037	39%	35%
		Ours	84.13	ℓ_1	6.887	8.859	2.541	2.747	37%	31%
				ℓ_2	0.529	0.655	0.408	0.448	77%	68%
				ℓ_∞	0.088	0.110	0.032	0.035	36%	32%
	2	Standard	83.23	ℓ_1	192.383	196.775	0.003	0.005	1.5E-5	2.4E-5
				ℓ_2	15.561	15.670	0.002	0.004	1.2E-4	2.3E-4
				ℓ_∞	2.258	2.590	0.000	0.001	1.2E-4	2.7E-4
		None	83.73	ℓ_1	6.759	8.214	1.671	1.707	25%	21%
				ℓ_2	0.516	0.616	0.285	0.292	55%	47%
				ℓ_∞	0.083	0.104	0.023	0.023	27%	22%
		Ours	83.34	ℓ_1	7.003	8.920	1.661	1.711	24%	19%
				ℓ_2	0.536	0.665	0.283	0.293	53%	44%
				ℓ_∞	0.089	0.114	0.022	0.023	25%	20%

Table 5: Clean accuracies, upper bounds, certified lower bounds by our method of models with different layer normalization settings.