POPQORN: Quantifying robustness of recurrent neural networks

POPQORN: Quantifying Robustness of Recurrent Neural Networks

Ching-Yun Ko [*1] **Zhaoyang Lyu** [*2] **Tsui-Wei Weng** [3] **Luca Daniel** [3] **Ngai Wong** [1] **Dahua Lin** [2]

Abstract

The vulnerability to adversarial attacks has been a critical issue for deep neural networks. Addressing this issue requires a reliable way to evaluate the robustness of a network. Recently, several methods have been developed to compute *robustness quantification* for neural networks, namely, certified lower bounds of the minimum adversarial perturbation. Such methods, however, were devised for feed-forward networks, *e.g.* multi-layer perceptron or convolutional networks. It remains an open problem to quantify robustness for recurrent networks, especially LSTM and GRU. For such networks, there exist additional challenges in computing the robustness quantification, such as handling the inputs at multiple steps and the interaction between gates and states. In this work, we propose *POPQORN* (**P**ropagated-**o**ut**p**ut **Q**uantified R**o**bustness for **RN**Ns), a general algorithm to quantify robustness of RNNs, including vanilla RNNs, LSTMs, and GRUs. We demonstrate its effectiveness on different network architectures and show that the robustness quantification on individual steps can lead to new insights.

1. Introduction

Deep learning has led to remarkable performance gains on a number of tasks, *i.e.* image classification, speech recognition, and language processing. Nevertheless, recent literature has demonstrated that adversarial examples broadly exist for deep neural networks in these applications (Szegedy et al., 2014; Kurakin et al., 2017; Jia & Liang, 2017; Carlini & Wagner, 2018). A small perturbation that humans are mostly immune to can be crafted to mislead a neural network's predictions. As deep neural networks have been widely employed in many safety-critical applications (Sharif et al., 2016; Kurakin et al., 2017; Eykholt et al., 2018), it is crucial to know when such models will fail and what can be done to make them more robust to the adversarial attacks.

Studies on the robustness of neural networks mainly fall in two categories: (1) *Attack-based approaches* – researchers try to design strong adversarial attack algorithms to attack deep neural networks and the robustness is measured by the distortion between successful adversarial examples and the original ones. (2) *Verification-based approaches* (Katz et al., 2017; Cheng et al., 2017), which aim to find the minimum distortion of neural networks or its lower bounds that are *agnostic to attack methods*. Existing verification-based methods were mainly devised for feed-forward networks, such as multi-layer perceptron. Robustness verification for recurrent neural networks (RNNs), which have been widely used in speech recognition and natural language processing, has not been systematically investigated.

To verify the robustness of RNNs, we face several new challenges: (1) Some popular RNN formulations, including LSTM and GRU, involve hidden states that are indirectly determined by three to four gates. These nonlinear gates are tightly coupled together, which we refer to as *cross-nonlinearity*. This substantially complicates the verification of robustness. (2) RNNs are widely adopted for applications with sequential inputs, *e.g.* sentences or time series. However, previous verification methods typically assume that the inputs were fed into the network at the bottom layer. Hence, they are not directly applicable here. (3) For applications with sequential inputs, imperceptible adversarial examples may correspond to texts with least number of changed words (Gao et al., 2018). Thus it is critical to evaluate the hardness of manipulating one single word (one input frame) instead of all words.

In this paper, we tackle the aforementioned problems by proposing an effective robustness quantification framework called POPQORN (**P**ropagated-**o**ut**p**ut **Q**uantified R**o**bustness for **RN**Ns) for RNNs. We bound the nonlinear activation function using linear functions. Starting from the output layer, linear bounds are propagated back to

[*]Equal contribution [1]The University of Hong Kong, Hong Kong [2]The Chinese University of Hong Kong, Hong Kong [3]Massachusetts Institute of Technology, Cambridge, MA, USA. Source code is available at https://github.com/ZhaoyangLyu/POPQORN. Correspondence to: Ching-Yun Ko <cyko@eee.hku.hk>, Zhaoyang Lyu <lyuzhaoyang@link.cuhk.edu.hk>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

Table 1. Comparison of methods for evaluating RNN robustness.

Method	Task	Application	Architecture	Attack	Verification	Robustness guarantee
FGSM (Papernot et al., 2016a)	Categorical/ Sequential	NLP	LSTM	✓	×	×
(Gong & Poellabauer, 2017)	Categorical	Speech	WaveRNN (RNN/ LSTM)	✓	×	×
Houdini (Cissé et al., 2017)	Sequential	Speech	DeepSpeech-2 (LSTM)	✓	×	×
(Jia & Liang, 2017)	SQuAD	NLP	LSTM	✓	×	×
(Zhao et al., 2018)	Categorical	NLP	LSTM	✓	×	×
(Ebrahimi et al., 2018a;b)	Categorical/ Sequential	NLP	LSTM	✓	×	×
Seq2Sick (Cheng et al., 2018)	Sequential	NLP	Seq2seq (LSTM)	✓	×	×
C&W (Carlini & Wagner, 2018)	Sequential	Speech	DeepSpeech (LSTM)	✓	×	×
CLEVER (Weng et al., 2018b)	Categorical	CV/ NLP/ Speech	RNN/ LSTM/ GRU	×	✓	×
POPQORN (This work)	Categorical	CV/ NLP/ Speech	RNN/ LSTM/ GRU	×	✓	✓

Table 2. Comparison of methods for providing adversarial robustness quantification in NNs.

Method	Certification	Multi-layer	Beyond ReLU/ MLP	RNN structures	handle cross-nonlinearity	Implementation
(Hein & Andriushchenko, 2017)	✓	×	differentiable/×	×	×	TensorFlow
CLEVER (Weng et al., 2018b)	×	✓	✓/ ✓	×	✓	NumPy
SDP approach (Raghunathan et al., 2018)	✓	×	✓/ ×	×	×	TensorFlow
Dual approach (Dvijotham et al., 2018)	✓	✓	✓/ ✓	×	×	Not specified
Fast-lin / Fast-lip (Weng et al., 2018a)	✓	✓	×/ ×	×	×	NumPy
CROWN (Zhang et al., 2018)	✓	✓	✓/ ×	×	×	NumPy
DeepZ (Singh et al., 2018)	✓	✓	✓/ no .pooling layers	×	×	Python, C
CNN-Cert (Boopathy et al., 2019)	✓	✓	✓/ ✓	×	×	NumPy
POPQORN (This work)	✓	✓	✓/ ✓	✓	✓	PyTorch (GPU)

the first layer recursively. Compared to existing methods, POPQORN has three important advantages: (1) *Novel* - it is a general framework, which is, to the best of our knowledge, the **first** work to provide a quantified robustness evaluation for RNNs with robustness guarantees. (2) *Effective* - it can handle complicated RNN structures besides vanilla RNNs, including LSTMs and GRUs that contain challenging coupled nonlinearities. (3) *Versatile* - it can be widely applied in applications including but not limited to computer vision, natural language processing, and speech recognition.

2. Background and Related Works

Adversarial attacks in RNNs. Crafting adversarial examples of RNNs in natural language processing and speech recognition has started to draw public attentions in addition to the adversarial examples of feed-forward networks in image classifications. Adversarial attacks of RNNs on text classification task (Papernot et al., 2016a), reading comprehension systems (Jia & Liang, 2017), seq2seq models (Cheng et al., 2018) have been proposed in natural language processing application; meanwhile recently (Gong & Poellabauer, 2017) and (Cissé et al., 2017; Carlini & Wagner, 2018) have also demonstrated successful attacks in speech recognition and audio systems. We summarize the RNN attacks in Table 1. (Papernot et al., 2016b; Gong & Poellabauer, 2017; Ebrahimi et al., 2018a;b; Cheng et al., 2018; Carlini & Wagner, 2018) perform gradient-based attacks, while generative adversarial network is used in (Zhao et al., 2018) to generate adversaries. We adapt (Carlini & Wagner, 2017) into C&W-Ada for finding RNN attacks.

Robustness verification for neural networks. To safeguard against misclassification under a threat model, *verification-based* methods evaluate the strengths such that any possible attacks weaker than the proposed strengths will fail. A commonly-used threat model is the norm-ball bounded attacks, wherein strengths of adversaries are quantified by their l_p distance from the original example. Under the norm-ball bounded threat model, determining the minimum adversarial distortion of ReLU networks has been shown to be NP-hard (Katz et al., 2017). Despite the hardness of the problem, fortunately, it is possible to estimate the minimum adversarial distortion (Weng et al., 2018b) or to compute a non-trivial certified lower bound (Hein & Andriushchenko, 2017; Raghunathan et al., 2018; Weng et al., 2018a; Dvijotham et al., 2018; Zhang et al., 2018; Singh et al., 2018; Boopathy et al., 2019). In (Weng et al., 2018b), the authors proposed a robustness score, CLEVER, for neural network image classifiers based on estimating local Lipschitz constant of the networks. We show that it is possible to directly adapt their framework and compute a CLEVER score for RNNs, which is referred to CLEVER-RNN in Section 4.

However, CLEVER score does not come with guarantees and therefore an alternative approach is to compute a non-trivial lower bound of the minimum adversarial distortion. (Hein & Andriushchenko, 2017) and (Raghunathan et al., 2018) proposed to analytically compute the lower bounds for shallow networks (with no more than 2 hidden layers) but their bounds get loose easily when applied to general networks. (Weng et al., 2018a) proposed two algorithms Fast-Lin and Fast-Lip to efficiently quantify robustness for ReLU networks and the Fast-Lin algorithm is further extended to CROWN (Zhang et al., 2018) to quantify for MLP networks with general activation functions. On the other hand, (Dvijotham et al., 2018) proposed a dual-approach to verify NNs with a general class of activation functions. However, their approach compromises the lower bound performance and

the computational efficiency of the "anytime" algorithm. Recently, (Boopathy et al., 2019) proposes a framework that is capable of quantifying robustness on CNNs with various architectures including pooling layers, residual blocks in contrast to (Singh et al., 2018) that is limited to convolutional architectures without pooling layers.

To the best of our knowledge, there is no prior work to provide robustness verification for RNNs that tackles the aforementioned challenges: the complex feed-back architectures, the sequential inputs, and the cross-nonlinearity of the hidden states. In line with the robustness verification for NNs, this paper is the first to study the robustness properties of RNNs (see Tables 1 and 2 for detailed comparisons). Since it is not straightforward to define a mistake in tasks other than classifications (cf. a mistake in classification tasks refers to a misclassification), we limit the scope of this paper to RNN-based classifiers.

3. POPQORN: Quantifying Robustness of Recurrent Neural Networks

Overview. In this section, we show that the output of an RNN can be bounded by two *linear* functions when the input sequence of the network is perturbed within an ℓ_p ball with a radius ϵ. By applying the *linear* bounds on the *non-linear* activation functions (e.g. sigmoid and tanh) and the *non-linear* operations (e.g. cell states in LSTM), we can *decouple* the non-linear activations and the *cross-nonlinearity* in the hidden states layer by layer and eventually bound the network output by two *linear* functions in terms of input[1]. Subsequently, we show how this theoretical result is used in the POPQORN to compute robustness quantification of an RNN. For ease of illustrations, we start with two motivating examples on a 2-layer vanilla RNN and a 1-layer LSTM network. The complete theorems for general m-layer vanilla RNNs, LSTMs, and GRUs are provided in Section A in the appendix.

Notations. Let $\mathbf{X_0} = [\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(m)}]$ be the original input data sequence, and let $\mathbf{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}]$ be the perturbed frames of $\mathbf{X_0}$ within an ϵ-bounded l_p-ball, i.e., $\mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$, where superscript "$(k)$" denotes the k-th time step. An RNN function is denoted as F and the j-th output element as $F_j(\mathbf{X})$. The upper and lower bounds of $F_j(\mathbf{X})$ are denoted as $F_j^U(\mathbf{X})$ and $F_j^L(\mathbf{X})$, respectively. The full notations are summarized in Table 3.

3.1. A 2-layer Vanilla RNN

Definition. A many-to-one 2-layer ($m = 2$) RNN reads

$$F(\mathbf{X}) = \mathbf{W}^{Fa}\mathbf{a}^{(2)} + \mathbf{b}^F,$$

[1]Proposed bounding techniques are applicable to any non-linear activation that is bounded above and below for the given interval.

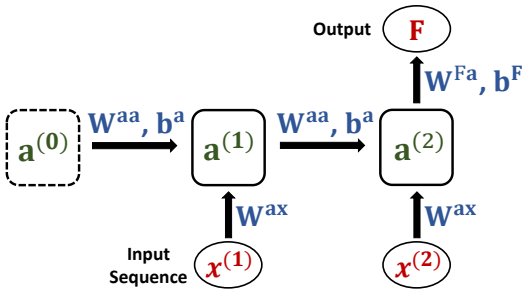


Figure 1. Graphical depiction of a 2-layer many-to-one RNN.

$$\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{aa}\mathbf{a}^{(1)} + \mathbf{W}^{ax}\mathbf{x}^{(2)} + \mathbf{b}^a),$$
$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}^{aa}\mathbf{a}^{(0)} + \mathbf{W}^{ax}\mathbf{x}^{(1)} + \mathbf{b}^a),$$

where F is the output, $\mathbf{a}^{(k)}$ is the k-th hidden state, $\sigma(\cdot)$ is the coordinate-wise activation function, and $\mathbf{W}^{aa}, \mathbf{W}^{Fa}, \mathbf{b}^F, \mathbf{b}^a$ are associated model parameters, as shown in Figure 1.

Ideas. Based on the above equation, for a fixed $j \in [t]$, we aim at deriving two explicit functions F_j^L and F_j^U such that $\forall\, \mathbf{X} \in \mathbb{R}^{n\times 2}$ where $\mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$, the inequality $F_j^L(\mathbf{X}) \le F_j(\mathbf{X}) \le F_j^U(\mathbf{X})$ holds true. To start with, we assume that we know the bounds for *pre-activation* $\mathbf{y}^{(k)} = \mathbf{W}^{aa}\mathbf{a}^{(k-1)} + \mathbf{W}^{ax}\mathbf{x}^{(k)} + \mathbf{b}^a$ (superscript "(k)" denotes the k-th layer): $\mathbf{l}^{(k)} \preccurlyeq \mathbf{y}^{(k)} \preccurlyeq \mathbf{u}^{(k)}$. We refer the $\mathbf{l}$ and $\mathbf{u}$ as *pre-activation bounds* in the remainder of this paper. We show that it is possible to bound every non-linear activation function $\sigma(\cdot)$ in the hidden states by two bounding lines in (1a) and (1b) associated to $\mathbf{l}^{(k)}$ and $\mathbf{u}^{(k)}$, and that we can apply this procedure recursively from the output $F_j(\mathbf{X})$ to the input $\mathbf{X}$ to obtain F_j^L and F_j^U.

Bounding lines. Two univariate bounding linear functions are defined as follows:

$$h_{U,r}^{(k)}(\mathbf{v}) = \alpha_{U,r}^{(k)}(\mathbf{v} + \beta_{U,r}^{(k)}), \tag{1a}$$
$$h_{L,r}^{(k)}(\mathbf{v}) = \alpha_{L,r}^{(k)}(\mathbf{v} + \beta_{L,r}^{(k)}), \tag{1b}$$

such that for $\mathbf{l}_r^{(k)} \le \mathbf{v} \le \mathbf{u}_r^{(k)}$,

$$\text{eq.(1b)} \le \sigma(\mathbf{v}) \le \text{eq.(1a)},$$

where the r in the subscript implies the dependency of the derived lines on neurons. Both the slopes and intercepts are functions of *pre-activation* bounds.

Derivation. We exemplify how a 2-layer vanilla RNN can be bounded:

$$F_j(\mathbf{X}) = \mathbf{W}_{j,:}^{Fa}\mathbf{a}^{(2)} + \mathbf{b}_j^F = \mathbf{W}_{j,:}^{Fa}\sigma(\mathbf{y}^{(2)}) + \mathbf{b}_j^F. \tag{2}$$

We use s upper-bounding lines $h_{U,r}^{(2)}(\mathbf{y}_r^{(2)}), r \in [s]$, and also use variables $\lambda_{j,r}^{(2)}$ and $\mathbf{\Delta}_{r,j}^{(2)}$ to denote the slopes in front of

Table 3. Table of Notation

Notation	Definition	Notation	Definition	Notation	Definition
t	number of output classes	$F : \mathbb{R}^{n\times m} \to \mathbb{R}^t$	network classifier	$\mathbf{X}_0 \in \mathbb{R}^{n\times m}$	original input
n	size of input frames	$\mathbf{x}^{(k)} \in \mathbb{R}^n$	the k-th input frame	$[K]$	set $\{1, 2, \cdots, K\}$
s	number of neurons in a layer	$F_j^L(\mathbf{X}) : \mathbb{R}^{n\times m} \to \mathbb{R}$	linear lower bound of $F_j(\mathbf{X})$	$\mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$	$\{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_0^{(k)}\|_p \le \epsilon\}$
m	number of network layers	$F_j^U(\mathbf{X}) : \mathbb{R}^{n\times m} \to \mathbb{R}$	linear upper bound of $F_j(\mathbf{X})$	$\mathbf{l} \preccurlyeq \mathbf{y} \preccurlyeq \mathbf{u}$	$\mathbf{l}_r \le \mathbf{y}_r \le \mathbf{u}_r$, $\forall\, r \in [s], \mathbf{l}, \mathbf{y}, \mathbf{u} \in \mathbb{R}^s$
$\mathbf{a}_0$	initial hidden state	γ_j^L	global lower bound of $F_j(\mathbf{X})$		
$\mathbf{c}_0$	initial cell state	γ_j^U	global upper bound of $F_j(\mathbf{X})$		

$\mathbf{y}_r^{(2)}$ and intercepts in the parentheses:

$$\lambda_{j,r}^{(2)} = \begin{cases} \alpha_{U,r}^{(2)} & \text{if } \mathbf{W}_{j,r}^{Fa} \ge 0; \\ \alpha_{L,r}^{(2)} & \text{if } \mathbf{W}_{j,r}^{Fa} < 0; \end{cases} \quad \mathbf{\Delta}_{r,j}^{(2)} = \begin{cases} \beta_{U,r}^{(2)} & \text{if } \mathbf{W}_{j,r}^{Fa} \ge 0; \\ \beta_{L,r}^{(2)} & \text{if } \mathbf{W}_{j,r}^{Fa} < 0, \end{cases}$$

and obtain

$$F_j(\mathbf{X}) \le (\mathbf{W}_{j,:}^{Fa} \odot \lambda_{j,:}^{(2)})(\mathbf{y}^{(2)} + \mathbf{\Delta}_{:,j}^{(2)}) + \mathbf{b}_j^F,$$

where $\odot$ is the Hadamard (i.e. element-wise) product. To simplify notation, we let $\mathbf{\Lambda}_{j,:}^{(2)} := \mathbf{W}_{j,:}^{Fa} \odot \lambda_{j,:}^{(2)}$ and have

$$F_j(\mathbf{X}) \le \tilde{\mathbf{W}}_{j,:}^{aa(2)}\mathbf{a}^{(1)} + \tilde{\mathbf{W}}_{j,:}^{ax(2)}\mathbf{x}^{(2)} + \tilde{\mathbf{b}}_j^{(2)},$$

where

$$\tilde{\mathbf{W}}_{j,:}^{aa(2)} = \mathbf{\Lambda}_{j,:}^{(2)}\mathbf{W}^{aa},\ \tilde{\mathbf{W}}_{j,:}^{ax(2)} = \mathbf{\Lambda}_{j,:}^{(2)}\mathbf{W}^{ax},$$
$$\tilde{\mathbf{b}}_j^{(2)} = \mathbf{\Lambda}_{j,:}^{(2)}(\mathbf{b}^a + \mathbf{\Delta}_{:,j}^{(2)}) + \mathbf{b}_j^F.$$

Substituting $\mathbf{a}^{(1)}$ with its definition yields

$$F_j(\mathbf{X}) \le \tilde{\mathbf{W}}_{j,:}^{aa(2)}\sigma(\mathbf{y}^{(1)}) + \tilde{\mathbf{W}}_{j,:}^{ax(2)}\mathbf{x}^{(2)} + \tilde{\mathbf{b}}_j^{(2)}. \quad (3)$$

Note that Equations (2) and (3) are in similar forms. Thus we can bound $\sigma(\mathbf{y}^{(1)})$ by s linear functions $h_{U,r}^{(1)}(\mathbf{y}_r^{(1)}), r \in [s]$ and use $\lambda_{j,r}^{(1)}$ and $\mathbf{\Delta}_{r,j}^{(1)}$ to denote slopes in front of $\mathbf{y}_r^{(1)}$ and intercepts in the parentheses:

$$\lambda_{j,r}^{(1)} = \begin{cases} \alpha_{U,r}^{(1)} & \text{if } \tilde{\mathbf{W}}_{j,r}^{aa(2)} \ge 0; \\ \alpha_{L,r}^{(1)} & \text{if } \tilde{\mathbf{W}}_{j,r}^{aa(2)} < 0; \end{cases}$$
$$\mathbf{\Delta}_{r,j}^{(1)} = \begin{cases} \beta_{U,r}^{(1)} & \text{if } \tilde{\mathbf{W}}_{j,r}^{aa(2)} \ge 0; \\ \beta_{L,r}^{(1)} & \text{if } \tilde{\mathbf{W}}_{j,r}^{aa(2)} < 0, \end{cases}$$

and let $\mathbf{\Lambda}_{j,:}^{(1)} := \tilde{\mathbf{W}}_{j,:}^{aa(2)} \odot \lambda_{j,:}^{(1)}$. Then we have

$$F_j(\mathbf{X}) \le \mathbf{\Lambda}_{j,:}^{(1)}\mathbf{W}^{aa}\mathbf{a}^{(0)} + \sum_{z=1}^{2}\mathbf{\Lambda}_{j,:}^{(z)}\mathbf{W}^{ax}\mathbf{x}^{(z)}$$
$$+ \sum_{z=1}^{2}\mathbf{\Lambda}_{j,:}^{(z)}(\mathbf{b}^a + \mathbf{\Delta}_{:,j}^{(z)}) + \mathbf{b}_j^F.$$

So far we have derived an explicit linear function on the right-hand side of the above inequality. We denote it herein as F_j^U and we have $F_j(\mathbf{X}) \le F_j^U(\mathbf{X}), \forall\, \mathbf{X} \in \mathbb{R}^{n\times 2}$ where $\mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$. A closed-form global upper bound γ_j^U can be obtained naturally by maximizing $F_j^U(\mathbf{X})$ for all possible $\mathbf{X}$, which can be directly solved by applying Holder's inequality. Hence we obtain

$$\gamma_j^U = \mathbf{\Lambda}_{j,:}^{(1)}\mathbf{W}^{aa}\mathbf{a}^{(0)} + \sum_{z=1}^{2}\epsilon\|\mathbf{\Lambda}_{j,:}^{(z)}\mathbf{W}^{ax}\|_q$$
$$+ \sum_{z=1}^{2}\mathbf{\Lambda}_{j,:}^{(z)}\mathbf{W}^{ax}\mathbf{x}_0^{(z)} + \sum_{z=1}^{2}\mathbf{\Lambda}_{j,:}^{(z)}(\mathbf{b}^a + \mathbf{\Delta}_{:,j}^{(z)}) + \mathbf{b}_j^F.$$

An explicit function F_j^L and a closed-form global lower bound γ_j^L can also be found through similar steps and by minimizing $F_j^L(\mathbf{X})$ instead. During the derivation, we have limited perturbations to be uniform across input frames, which assemble noises in cameras. However, the above method is also applicable to non-uniform distortions. For example, we can also certify bounds for distortions on parts of the input frames, which on the other hand are of more interests in natural language processing tasks.

3.2. A 1-layer Long Short-term Memory Network

For an RNN with LSTM units, we can also derive analytic upper-bounding and lower-bounding functions. In this example, we inherit the notations in Table 3, and exemplify through a 1-layer ($m = 1$) LSTM as shown in Figure 2.

LSTM. The following updating equations are adopted:

$$\text{Input gate: } \mathbf{i}^{(k)} = \sigma(\mathbf{W}^{ix}\mathbf{x}^{(k)} + \mathbf{W}^{ia}\mathbf{a}^{(k-1)} + \mathbf{b}^i);$$
$$\text{Forget gate: } \mathbf{f}^{(k)} = \sigma(\mathbf{W}^{fx}\mathbf{x}^{(k)} + \mathbf{W}^{fa}\mathbf{a}^{(k-1)} + \mathbf{b}^f);$$
$$\text{Cell gate: } \mathbf{g}^{(k)} = \tanh(\mathbf{W}^{gx}\mathbf{x}^{(k)} + \mathbf{W}^{ga}\mathbf{a}^{(k-1)} + \mathbf{b}^g);$$
$$\text{Output gate: } \mathbf{o}^{(k)} = \sigma(\mathbf{W}^{ox}\mathbf{x}^{(k)} + \mathbf{W}^{oa}\mathbf{a}^{(k-1)} + \mathbf{b}^o);$$
$$\text{Cell state: } \mathbf{c}^{(k)} = \mathbf{f}^{(k)} \odot \mathbf{c}^{(k-1)} + \mathbf{i}^{(k)} \odot \mathbf{g}^{(k)};$$
$$\text{Hidden state: } \mathbf{a}^{(k)} = \mathbf{o}^{(k)} \odot \tanh(\mathbf{c}^{(k)}).$$

Again, the $\sigma(\cdot)$ denotes the coordinate-wise sigmoid function and $\tanh(\cdot)$ is the coordinate-wise hyperbolic tangent function. Output F of the network is determined by $F(\mathbf{X}) = \mathbf{W}^{Fa}\mathbf{a}^{(m)} + \mathbf{b}^F$, and we have pre-activations as

$$\mathbf{y}^{i(k)} = \mathbf{W}^{ix}\mathbf{x}^{(k)} + \mathbf{W}^{ia}\mathbf{a}^{(k-1)} + \mathbf{b}^i;$$
$$\mathbf{y}^{f(k)} = \mathbf{W}^{fx}\mathbf{x}^{(k)} + \mathbf{W}^{fa}\mathbf{a}^{(k-1)} + \mathbf{b}^f;$$
$$\mathbf{y}^{g(k)} = \mathbf{W}^{gx}\mathbf{x}^{(k)} + \mathbf{W}^{ga}\mathbf{a}^{(k-1)} + \mathbf{b}^g;$$
$$\mathbf{y}^{o(k)} = \mathbf{W}^{ox}\mathbf{x}^{(k)} + \mathbf{W}^{oa}\mathbf{a}^{(k-1)} + \mathbf{b}^o.$$

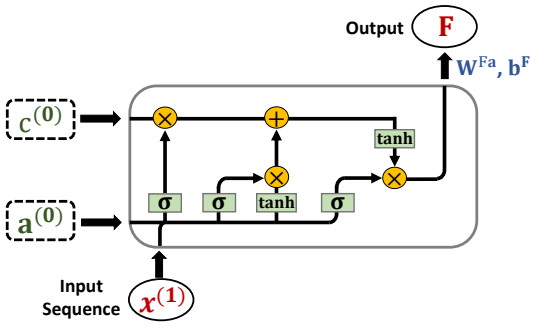


Figure 2. Graphical depiction of a 1-layer LSTM network

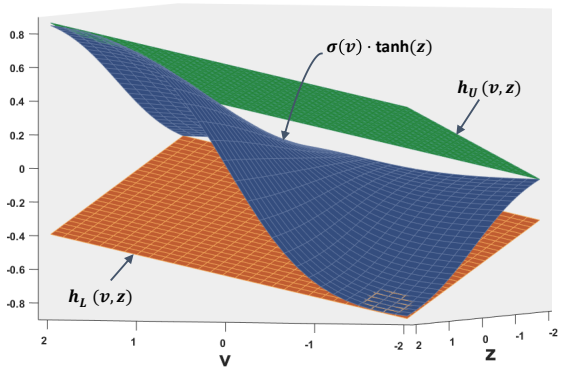


Figure 3. Illustration of the upper-bounding plane $h_U(\mathbf{v}, \mathbf{z})$ and lower-bounding plane $h_L(\mathbf{v}, \mathbf{z})$ of $\sigma(\mathbf{v})\tanh(\mathbf{z})$.

Ideas. When deriving the lower bounds and upper bounds of LSTMs, we have also made an important assumption: we assume we know the bounds for hidden/cell states. This assumption, as well as the assumption made when deriving for vanilla RNNs and GRUs, can be easily fulfilled as the bounds of hidden/cell states are available by similar derivations as will be shown below. We refer readers to Theorem A.2 (vanilla RNNs), Corollary A.3/ Theorem A.4 (LSTMs) and Corollary A.6 (GRUs) in the appendix for details.

Recalling that in Section 3.1, we aim at bounding the non-linearity using univariate linear functions. Final bounds are obtained by recursively propagating the linear bounds from the output layer back to the first layer. Here when analyzing the bounds for an LSTM, we adopt a similar approach of propagating linear bounds from the last layer to the first. However, different from the vanilla RNN, the difficulty of reaching this goal lies in bounding more complex non-linearities. In an LSTM, we cope with two different non-linear functions:

$$\sigma(\mathbf{v})\mathbf{z} \quad \text{and} \quad \sigma(\mathbf{v})\tanh(\mathbf{z}),$$

both of which are dependent on two variables and are cross terms of varied gates. To deal with this, we extend our previous ideas to using *planes* instead of *lines* to bound these cross terms. The graphical illustration of the bounding planes is shown in Figure 3.

Bounding planes. With bounds of hidden/cell states, we can gather bounds: $\mathbf{l}^{\text{gate}(1)} \preccurlyeq \mathbf{y}^{\text{gate}(1)} \preccurlyeq \mathbf{u}^{\text{gate}(1)}$ where gate $= \{i, f, g, o\}$ and superscript "(1)" denotes the layer we are at. As both variables are bounded as above and compactness is a continuous invariant, we are guaranteed to have at least two bounding planes for the non-linearities. For example, for $\sigma(\mathbf{v})\mathbf{z}$ we have planes:

$$h_{U,r}^{\text{cross}(1)} = \max(\sigma(\mathbf{v})\mathbf{z}), \quad h_{L,r}^{\text{cross}(1)} = \min(\sigma(\mathbf{v})\mathbf{z}).$$

In practice, we use the following planes:

$$h_{U,r}^{\text{cross}(1)}(\mathbf{v}, \mathbf{z}) = \alpha_{U,r}^{\text{cross}(1)}\mathbf{v} + \beta_{U,r}^{\text{cross}(1)}\mathbf{z} + \gamma_{U,r}^{\text{cross}(1)}, \quad \text{(4a)}$$

$$h_{L,r}^{\text{cross}(1)}(\mathbf{v}, \mathbf{z}) = \alpha_{L,r}^{\text{cross}(1)}\mathbf{v} + \beta_{L,r}^{\text{cross}(1)}\mathbf{z} + \gamma_{L,r}^{\text{cross}(1)}, \quad \text{(4b)}$$

that satisfy

$$\text{eq.(4b)} \leq \sigma(\mathbf{v})\mathbf{z} \leq \text{eq.(4a)},$$

where the r in the subscript implies the dependency of the derived planes on neurons, and superscript cross$\in \{ig, oc, fc\}$ tracks the origins of those cross terms. For example, cross$= ig$ when it is the coupling of input gates and cell gates: $\mathbf{v} = \mathbf{y}^{i(1)}$ and $\mathbf{z} = \mathbf{y}^{g(1)}$. Notably, slopes and intercepts, say $\alpha_{U,r}^{ig(1)}$ depends on ranges of $\mathbf{y}_r^{i(1)}$ and $\mathbf{y}_r^{g(1)}$. We formulate the task of finding bounding planes as a constrained optimization problem and use gradient descent method to solve it (see Section A.4 in the appendix for details).

Derivation. We now exemplify how a 1-layer LSTM can be bounded:

$$\begin{aligned} F_j(\mathbf{X}) &= \mathbf{W}_{j,:}^{Fa}\mathbf{a}^{(1)} + \mathbf{b}_j^F, \\ &= \mathbf{W}_{j,:}^{Fa}[\sigma(\mathbf{y}^{o(1)}) \odot \tanh(\mathbf{c}^{(1)})] + \mathbf{b}_j^F. \end{aligned} \quad (5)$$

To bound Equation (5), we use s upper-bounding planes $h_{U,r}^{oc(1)}(\mathbf{y}^{o(1)}, \mathbf{c}^{(1)}), r \in [s]$, and also define $\lambda_{j,r}^{oc(1)}$, $\mathbf{\Delta}_{j,r}^{oc(1)}$ and $\varphi_{j,r}^{oc(1)}$ in the parentheses:

$$\lambda_{j,r}^{oc(1)} = \begin{cases} \alpha_{U,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} \geq 0; \\ \alpha_{L,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} < 0; \end{cases}$$

$$\mathbf{\Delta}_{j,r}^{oc(1)} = \begin{cases} \beta_{U,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} \geq 0; \\ \beta_{L,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} < 0; \end{cases}$$

$$\varphi_{j,r}^{oc(1)} = \begin{cases} \gamma_{U,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} \geq 0; \\ \gamma_{L,r}^{oc(1)} & \text{if } \mathbf{W}_{j,r}^{Fa} < 0; \end{cases}$$

and obtain

$$\begin{aligned} F_j(\mathbf{X}) \leq\ & (\mathbf{W}_{j,:}^{Fa} \odot \lambda_{j,:}^{oc(1)})\mathbf{y}^{o(1)} + (\mathbf{W}_{j,:}^{Fa} \odot \mathbf{\Delta}_{j,:}^{oc(1)})\mathbf{c}^{(1)} \\ & + \sum_{r=1}^{s}(\mathbf{W}_{j,r}^{Fa}\varphi_{j,r}^{oc(1)}) + \mathbf{b}_j^F. \end{aligned}$$

Table 4. Quantified robustness bounds for various RNNs.

Networks	$\gamma_j^L \le F_j \le \gamma_j^U$	Closed-form formulas
Vanilla RNN	Upper bounds γ_j^U	$\mathbf{\Lambda}^{(0)}_{j,:}\mathbf{a}^{(0)} + \sum_{k=1}^m \epsilon\|\mathbf{\Lambda}^{(k)}_{j,:}\mathbf{W}^{ax}\|_q + \sum_{k=1}^m \mathbf{\Lambda}^{(k)}_{j,:}\mathbf{W}^{ax}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \mathbf{\Lambda}^{(k)}_{j,:}(\mathbf{b}^a + \mathbf{\Delta}^{(k)}_{:,j}) + \mathbf{b}_j^F$
	Lower bound γ_j^L	$\mathbf{\Omega}^{(0)}_{j,:}\mathbf{a}^{(0)} - \sum_{k=1}^m \epsilon\|\mathbf{\Omega}^{(k)}_{j,:}\mathbf{W}^{ax}\|_q + \sum_{k=1}^m \mathbf{\Omega}^{(k)}_{j,:}\mathbf{W}^{ax}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \mathbf{\Omega}^{(k)}_{j,:}(\mathbf{b}^a + \mathbf{\Theta}^{(k)}_{:,j}) + \mathbf{b}_j^F$
LSTM	Upper bounds γ_j^U	$\tilde{\mathbf{W}}^{a(1)}_{U,j,:}\mathbf{a}^{(0)} + \mathbf{\Lambda}^{fc(1)}_{\Delta,j,:}\mathbf{c}^{(0)} + \sum_{k=1}^m \epsilon\|\tilde{\mathbf{W}}^{x(k)}_{U,j,:}\|_q + \sum_{k=1}^m \tilde{\mathbf{W}}^{x(k)}_{U,j,:}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \tilde{\mathbf{b}}^{(k)}_{U,j} + \mathbf{b}_j^F$
	Lower bound γ_j^L	$\tilde{\mathbf{W}}^{a(1)}_{L,j,:}\mathbf{a}^{(0)} + \mathbf{\Omega}^{fc(1)}_{\Theta,j,:}\mathbf{c}^{(0)} - \sum_{k=1}^m \epsilon\|\tilde{\mathbf{W}}^{x(k)}_{L,j,:}\|_q + \sum_{k=1}^m \tilde{\mathbf{W}}^{x(k)}_{L,j,:}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \tilde{\mathbf{b}}^{(k)}_{L,j} + \mathbf{b}_j^F$
GRU	Upper bounds γ_j^U	$\tilde{\mathbf{W}}^{a(1)}_{U,j,:}\mathbf{a}^{(0)} + \sum_{k=1}^m \epsilon\|\tilde{\mathbf{W}}^{x(k)}_{U,j,:}\|_q + \sum_{k=1}^m \tilde{\mathbf{W}}^{x(k)}_{U,j,:}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \tilde{\mathbf{b}}^{(k)}_{U,j} + \mathbf{b}_j^F$
	Lower bound γ_j^L	$\tilde{\mathbf{W}}^{a(1)}_{L,j,:}\mathbf{a}^{(0)} - \sum_{k=1}^m \epsilon\|\tilde{\mathbf{W}}^{x(k)}_{L,j,:}\|_q + \sum_{k=1}^m \tilde{\mathbf{W}}^{x(k)}_{L,j,:}\mathbf{x}_0^{(k)} + \sum_{k=1}^m \tilde{\mathbf{b}}^{(k)}_{L,j} + \mathbf{b}_j^F$

Remark: see Section A in the appendix for detailed proofs and definitions.

For simplicity, we further collect the summing weights of $\mathbf{y}^{o(1)}$ and $\mathbf{c}^{(1)}$ into row vectors $\mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}$ and $\mathbf{\Lambda}^{oc(1)}_{\Delta,j,:}$, and constants (excepts $\mathbf{b}_j^F$) into $\mathbf{\Lambda}^{oc(1)}_{\varphi,j,r}$. Thereafter, we have

$$\begin{aligned} &F_j(\mathbf{X}) \\ &\le (\mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{ox})\mathbf{x}^{(1)} + (\mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{oa})\mathbf{a}^{(0)} + \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{b}^o \\ &+ \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:}[(\sigma(\mathbf{y}^{f(1)}) \odot \mathbf{c}^{(0)} + \sigma(\mathbf{y}^{i(1)}) \odot \tanh(\mathbf{y}^{g(1)})] \\ &+ \sum_{r=1}^{s} \mathbf{\Lambda}^{oc(1)}_{\varphi,j,r} + \mathbf{b}_j^F. \end{aligned} \tag{6}$$

Then to bound the two cross terms in Equation (6), we use upper-bounding planes $h^{fc(1)}_{U,r}(\mathbf{y}^{f(1)}, \mathbf{c}^{(0)})$ and $h^{ig(1)}_{U,r}(\mathbf{y}^{i(1)}, \mathbf{y}^{g(1)}), r \in [s]$. We define $\lambda^{\text{cross}'(1)}_{j,r}$, $\mathbf{\Delta}^{\text{cross}'(1)}_{j,r}$, $\varphi^{\text{cross}'(1)}_{j,r}$ for cross$' = \{fc, ig\}$ as in the parentheses:

$$\lambda^{\text{cross}'(1)}_{j,r} = \begin{cases} \alpha^{\text{cross}'(1)}_{U,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} \ge 0; \\ \alpha^{\text{cross}'(1)}_{L,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} < 0; \end{cases}$$

$$\mathbf{\Delta}^{\text{cross}'(1)}_{j,r} = \begin{cases} \beta^{\text{cross}'(1)}_{U,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} \ge 0; \\ \beta^{\text{cross}'(1)}_{L,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} < 0; \end{cases}$$

$$\varphi^{\text{cross}'(1)}_{j,r} = \begin{cases} \gamma^{\text{cross}'(1)}_{U,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} \ge 0; \\ \gamma^{\text{cross}'(1)}_{L,r} & \text{if } \mathbf{\Lambda}^{oc(1)}_{\Delta,j,r} < 0; \end{cases}$$

and obtain

$$\begin{aligned} F_j(\mathbf{X}) \le\ &\mathbf{p}_x\mathbf{x}^{(1)} + \mathbf{p}_a\mathbf{a}^{(0)} + \mathbf{\Lambda}^{fc(1)}_{\lambda,j,:}\mathbf{y}^{f(1)} + \mathbf{\Lambda}^{fc(1)}_{\Delta,j,:}\mathbf{c}^{(0)} \\ &+ \mathbf{\Lambda}^{ig(1)}_{\lambda,j,:}\mathbf{y}^{i(1)} + \mathbf{\Lambda}^{ig(1)}_{\Delta,j,:}\mathbf{y}^{g(1)} + \mathbf{p}_b + \mathbf{b}_j^F. \end{aligned}$$

where

$$\begin{aligned} \mathbf{p}_x &= \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{ox}, & \mathbf{p}_a &= \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{oa}, \\ \mathbf{\Lambda}^{fc(1)}_{\lambda,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \lambda^{fc(1)}_{j,:}, & \mathbf{\Lambda}^{fc(1)}_{\Delta,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \mathbf{\Delta}^{fc(1)}_{j,:}, \\ \mathbf{\Lambda}^{ig(1)}_{\lambda,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \lambda^{ig(1)}_{j,:}, & \mathbf{\Lambda}^{ig(1)}_{\Delta,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \mathbf{\Delta}^{ig(1)}_{j,:}, \\ \mathbf{\Lambda}^{fc(1)}_{\varphi,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \varphi^{fc(1)}_{j,:}, & \mathbf{\Lambda}^{ig(1)}_{\varphi,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\Delta,j,:} \odot \varphi^{ig(1)}_{j,:}, \\ \mathbf{p}_b &= \sum_{r=1}^{s}(\mathbf{\Lambda}^{fc(1)}_{\varphi,j,r} + \mathbf{\Lambda}^{ig(1)}_{\varphi,j,r} + \mathbf{\Lambda}^{oc(1)}_{\varphi,j,r}) + \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{b}^o. \end{aligned}$$

Substituting $\mathbf{y}^{f(1)}, \mathbf{c}^{(0)}, \mathbf{y}^{i(1)}, \mathbf{y}^{g(1)}$ with their definitions renders

$$F_j(\mathbf{X}) \le \tilde{\mathbf{W}}^{x(1)}_{U,j,:}\mathbf{x}^{(1)} + \tilde{\mathbf{W}}^{a(1)}_{U,j,:}\mathbf{a}^{(0)} + \tilde{\mathbf{b}}^{(1)}_{U,j} + \mathbf{\Lambda}^{fc(1)}_{\Delta,j,:}\mathbf{c}^{(0)},$$

where

$$\begin{aligned} \tilde{\mathbf{W}}^{x(1)}_{U,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{ox} + \mathbf{\Lambda}^{fc(1)}_{\lambda,j,:}\mathbf{W}^{fx} + \mathbf{\Lambda}^{ig(1)}_{\lambda,j,:}\mathbf{W}^{ix} \\ &\quad + \mathbf{\Lambda}^{ig(1)}_{\Delta,j,:}\mathbf{W}^{gx}, \\ \tilde{\mathbf{W}}^{a(1)}_{U,j,:} &= \mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{W}^{oa} + \mathbf{\Lambda}^{fc(1)}_{\lambda,j,:}\mathbf{W}^{fa} + \mathbf{\Lambda}^{ig(1)}_{\lambda,j,:}\mathbf{W}^{ia} \\ &\quad + \mathbf{\Lambda}^{ig(1)}_{\Delta,j,:}\mathbf{W}^{ga}, \\ \tilde{\mathbf{b}}^{(1)}_{U,j} &= [\mathbf{\Lambda}^{oc(1)}_{\lambda,j,:}\mathbf{b}^o + \mathbf{\Lambda}^{fc(1)}_{\lambda,j,:}\mathbf{b}^f + \mathbf{\Lambda}^{ig(1)}_{\lambda,j,:}\mathbf{b}^i + \mathbf{\Lambda}^{ig(1)}_{\Delta,j,:}\mathbf{b}^g \\ &\quad + \sum_{r=1}^{s}(\mathbf{\Lambda}^{oc(1)}_{\varphi,j,r} + \mathbf{\Lambda}^{fc(1)}_{\varphi,j,r} + \mathbf{\Lambda}^{ig(1)}_{\varphi,j,r})] + \mathbf{b}_j^F. \end{aligned}$$

A closed-form global upper bound γ_j^U can thereafter be obtained by applying Holder's inequality:

$$\begin{aligned} \gamma_j^U = \ &\epsilon\|\tilde{\mathbf{W}}^{x(1)}_{U,j,:}\|_q + \tilde{\mathbf{W}}^{x(1)}_{U,j,:}\mathbf{x}_0^{(1)} + \tilde{\mathbf{W}}^{a(1)}_{U,j,:}\mathbf{a}^{(0)} + \tilde{\mathbf{b}}^{(z)}_{U,j} \\ &+ \mathbf{\Lambda}^{fc(1)}_{\Delta,j,:}\mathbf{c}^{(0)}. \end{aligned}$$

An explicit function F_j^L and a closed-form global lower bound γ_j^L can also be found through similar steps such that $\gamma_j^L \le F_j^L(\mathbf{X}) \le F_j(\mathbf{X}), \forall\ \mathbf{X} \in \mathbb{R}^{n\times 2}$ where $\mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$. The derivations for a GRU follow similarly, and are put into the appendix (Section A.7) due to space constraint.

3.3. Robustness Quantification Algorithm

As summarized in Table 4, given a trained vanilla RNN, LSTM or GRU, input sequence $\mathbf{X}_0 \in \mathbb{R}^{n\times m}$, l_p ball parameters $p \ge 1$ and $\epsilon \ge 0$, for $\forall\ j \in [t]$, $1/q = 1 - 1/p$, there exist two fixed values γ_j^L and γ_j^U such that $\forall\ \mathbf{X} \in \mathbb{R}^{n\times m}$ where $\mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon)$, the inequality $\gamma_j^L \le F_j(\mathbf{X}) \le \gamma_j^U$ holds true. Now suppose the label of the input sequence is j, we aim at utilizing the uniform global bounds in Table 4 to find the largest possible lower bound ϵ_j of untargeted attacks. We formalize the objective and constraints as follows:

$$\epsilon_j = \max_{\epsilon} \epsilon,\ \text{s.t. } \gamma_j^L(\epsilon) \ge \gamma_i^U(\epsilon),\ \forall i \ne j.$$

To verify the largest possible lower bound $\hat{\epsilon}$ of targeted attacks (target class be i), we solve the following:

$$\hat{\epsilon}(i,j) = \max_{\epsilon} \epsilon,\ \text{s.t. } \gamma_j^L(\epsilon) \ge \gamma_i^U(\epsilon).$$

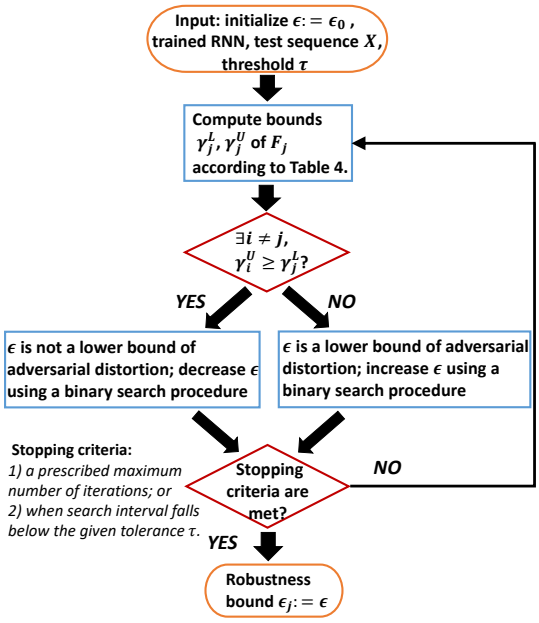


Figure 4. Steps in computing bounds for RNN networks.

One can then verify that $\epsilon_j = \min_{i \neq j} \hat{\epsilon}(i, j)$. Recalling from the derivations in Sections 3.1 and 3.2, slopes and intercepts utilized are functions of the input ranges. Therefore, the global lower bounds γ^L and upper bounds γ^U are also implicitly dependent on ϵ, preventing the above optimization problems from having an analytic solution. In this work, we conduct binary search procedures to compute the largest possible ϵ_j ($\hat{\epsilon}$). In practice, we start by initializing the ϵ to be $\epsilon_0 > 0$ and deriving upper and lower bounds according to Table 4. These are followed by a condition check that governs the next step. If $\exists\ i \neq j$ such that $\gamma_i^U \geq \gamma_j^L$, then we decrease ϵ; otherwise, we increase ϵ. We repeat the above procedure until a predefined maximum iterations are reached or when convergence is achieved.[2] Proposed algorithms are summarized in Section A.2 and A.5 in the appendix.

4. Experiments

Methods. The proposed POPQORN is used to quantify robustness for all models herein. To the best of our knowledge, there is **no** previous work done on quantifying robustness with guarantees for RNNs. Therefore, we compare our results with CLEVER score (Weng et al., 2018b) (an estimation of the minimum adversarial distortion) and C&W attack (Carlini & Wagner, 2017) (an upper bound of the minimum adversarial distortion). We emphasize on analyzing the characteristics of the certified bounds obtained by POPQORN and new insights they lead to. For comparison, we adapt CLEVER score to accommodate sequential input structure and denote it as CLEVER-RNN. Specifically, let j and i be the true and target labels, respectively. Assume $g(\mathbf{X}) = F_j(\mathbf{X}) - F_i(\mathbf{X})$ is a Lipschitz function, and define $L_q^t = \max\{\|\nabla_t g(\mathbf{X})\|_q : \mathbf{x}^{(k)} \in \mathbb{B}_p(\mathbf{x}_0^{(k)}, \epsilon_0), \forall k \in [m]\}$ where $\nabla_t g(\mathbf{X}) = (\frac{\partial g(\mathbf{X})}{\partial \mathbf{x}_1^{(t)}}, \ldots, \frac{\partial g(\mathbf{X})}{\partial \mathbf{x}_n^{(t)}})^T$. CLEVER-RNN score is then given as $\min\{\frac{g(\mathbf{X}_0)}{\sum_{t=1}^m L_q^t}, \epsilon_0\}$[3]. We also adapt C&W attack for our task and denote it as C&W-Ada[4]. The adapted C&W-Ada puts higher weights on finding a successful attack, and prioritizes the minimization of distortion magnitude when an attack is found. We use the maximum perturbation of all frames as the C&W score. We implement our algorithm using PyTorch to enable the use of GPUs. Using a server consisting of 16 NVIDIA Tesla K80 GPUs, it took about 4 hours to calculate certified bounds for 1000 samples in LSTMs with 4 frames, and one day in LSTMs with 14 frames. Except quantifying for LSTMs, the remaining experiments could be finished in about 10 hours using a single NVIDIA TITAN X GPU. More experimental details are given in the appendix Section B.

Experiment I. In this experiment, we evaluate POPQORN and other baselines on totally 12 vanilla RNNs and LSTMs trained on the MNIST dataset. Table 5 gives the certified lower bounds of models found by POPQORN, together with uncertified CLEVER-RNN scores, and upper bounds found by C&W-Ada attacks. Bounds obtained by POPQORN, CLEVER-RNN, C&W-Ada increase with numbers of hidden neurons s in RNNs and LSTMs, and generally decrease as the number of network layers m grows[5]. The uncertified bounds computed by CLEVER-RNN are similar to those found by C&W-Ada attacks, yet it should be noted that these bounds are without guarantees. A supplementary comparison among bounding techniques (2D bounding planes, 1D bounding lines, constants) for LSTM evaluations is provided in Appendix Section B.1 (accompanied with theorems in Section A.6).

Experiment II. Next, we evaluate the proposed POPQORN quantification on LSTMs trained on the MNIST sequence dataset[6]. Specifically, we compute the POPQORN bound on only one single input frame (i.e. we fix all input frames but one and derive certified bound for distortions on that frame). After calculating bounds of all input frames, we identify the frames with minimal bounds and call them *sensitive strokes*. In both subfigures of Figure 5, each point

[2]The binary search converges when the interval falls below the given tolerance.

[3]CLEVER score is defined by $\min\{\frac{g(\mathbf{X}_0)}{L_q}, \epsilon_0\}$, where $L_q = \max\{\|(\nabla_1 g(\mathbf{X}), \ldots, \nabla_m g(\mathbf{X}))\|_q : \mathbf{X} \in \mathbb{B}_p(\mathbf{X}_0, \epsilon_0)\}$. We refer readers to the appendix Sec. B.2 for the details.

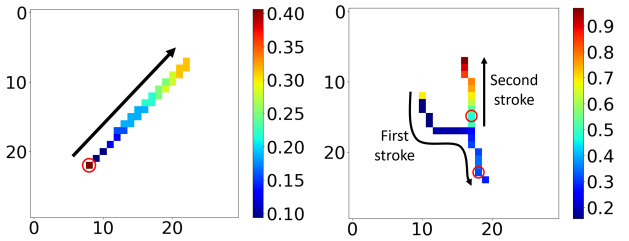
[4]We refer readers to the appendix Sec. B.3 for the details.

[5]The provable safety region (l_p balls) provided by POPQORN are distributed across framelets. The overall distortion allowed for an input sample is computed by $m^{1/p}\epsilon$.

[6]Freely available at https://edwin-de-jong.github.io/blog/mnist-sequence-data/

Table 5. (Experiment I) Averaged bounds and standard deviations (·/·) of POPCQRN and other baselines on MNIST dataset. POPQORN is the proposed method, CLEVER-RNN and C&W-Ada are adapted from (Weng et al., 2018b) and (Carlini & Wagner, 2017), respectively.

Network	l_p norm	Certified (POPQORN)	Uncertified (CLEVER-RNN)	Attack (C&W-Ada)	Network	l_p norm	Certified (POPQORN)	Uncertified (CLEVER-RNN)	Attack (C&W-Ada)
RNN	l_∞	0.0190/0.0047	0.0831/0.0398	0.2561/0.1372	RNN	l_∞	0.0087/0.0018	0.1259/0.0594	0.3267/0.1804
4 layers	l_2	0.2026/0.0680	0.8860/0.3920	1.3768/0.6076	14 layers	l_2	0.0526/0.0119	0.6864/0.3172	0.9278/0.4218
32 hidden nodes	l_1	1.0551/0.2689	5.0033/2.6034	5.1592/2.4726	64 hidden nodes	l_1	0.1578/0.0328	2.6018/1.3142	2.1149/0.9671
RNN	l_∞	0.0219/0.0047	0.1099/0.0503	0.2957/0.1374	LSTM	l_∞	0.0202/0.0055	0.0765/0.0356	0.2546/0.1202
4 layers	l_2	0.2487/0.0592	1.2006/0.5240	1.7072/0.6757	4 layers	l_2	0.2321/0.0636	0.8302/0.3714	1.3321/0.5529
64 hidden nodes	l_1	1.3199/0.2984	6.4128/3.0360	6.9347/3.1543	32 hidden nodes	l_1	1.1913/0.3385	4.6858/2.2761	4.9689/2.2740
RNN	l_∞	0.0243/0.0050	0.1405/0.0638	0.3537/0.1469	LSTM	l_∞	0.0218/0.0052	0.1032/0.0470	0.2893/0.1260
4 layers	l_2	0.2818/0.0557	1.4698/0.6595	2.1666/0.8556	4 layers	l_2	0.2448/0.0618	1.0947/0.4403	1.6888/0.6705
128 hidden nodes	l_1	1.4362/0.2880	7.7504/3.6968	8.8906/4.2051	64 hidden nodes	l_1	1.2644/0.3250	6.0216/2.8502	6.6931/2.9917
RNN	l_∞	0.0131/0.0031	0.0841/0.0421	0.2424/0.1291	LSTM	l_∞	0.0218/0.0044	0.1227/0.0508	0.3303/0.1358
7 layers	l_2	0.1045/0.0371	0.6846/0.3283	1.0300/0.4527	4 layers	l_2	0.2491/0.0523	1.3225/0.5412	1.9379/0.7593
32 hidden nodes	l_1	0.4492/0.1088	3.2046/1.7024	2.9425/1.4266	128 hidden nodes	l_1	1.2839/0.2760	6.8665/3.0725	7.7319/3.2369
RNN	l_∞	0.0131/0.0028	0.1013/0.0525	0.2660/0.1330	LSTM	l_∞	0.0165/0.0044	0.0924/0.0446	0.2635/0.1290
7 layers	l_2	0.1113/0.0246	0.8121/0.4210	1.1454/0.4931	7 layers	l_2	0.1400/0.0369	0.7287/0.3434	1.0733/0.4694
64 hidden nodes	l_1	0.4499/0.0951	3.5476/1.8650	3.4262/1.6191	32 hidden nodes	l_1	0.5680/0.1563	3.2266/1.7095	3.2309/1.4733
RNN	l_∞	0.0083/0.0023	0.0931/0.0506	0.2891/0.2004	LSTM	l_∞	0.0099/0.0028	0.1220/0.0676	0.3148/0.1700
14 layers	l_2	0.0493/0.0139	0.5074/0.2538	0.7577/0.3859	14 layers	l_2	0.0593/0.0148	0.6688/0.3346	0.9088/0.3854
32 hidden nodes	l_1	0.1459/0.0399	1.9323/1.0724	1.6339/0.8493	32 hidden nodes	l_1	0.1805/0.0496	2.4880/1.3743	2.0035/0.9447



(a) digit "1" (b) digit "4"

Figure 5. (Experiment II) Heat maps of certified 2-norm bounds computed by POPQORN. Left (a): an example of digit "1". Right (b): an example of digit "4". Arrows indicate the order of strokes.

records the relative displacement from the previous point. Heat maps are used to track the changes in sensitivities (quantified robustness scores) with strokes . We identify the starting point in Figure 5(a) with a circle. Notably, this point has a relatively big certified bound. This implies the loose connection between the starting point of one's stroke and the number to be written down. Another tendency we observe is that points in the back have larger bounds compared with points in the front. Since the position of a point only affects the points behind it, points in the back have less influence on the overall shape. Therefore, they can tolerate perturbations of larger magnitude without influencing its classification result. In Figure 5(b), we circle two points that are near the end of the first stroke and the start of the second stroke, respectively. These points have more influence on the overall shape of the sequence, which is also supported by the comparatively small POPQORN bounds.

Experiment III: Lastly, POPQORN is evaluated on LSTM classifiers for the question classification (UIUC's CogComp QC) Dataset (Li & Roth, 2002)[7]. Figure 6 shows two sample sentences in the UIUC's CogComp QC dataset. We conduct POPQORN quantification on individual steps

[7]Freely available at http://cogcomp.org/Data/QA/QC/

Example	What	is	the	name	of	Roy	Roger	's	dog	?
ENTY	0.34	0.50	0.53	0.27	0.39	0.19	0.32	1.02	0.67	0.93

Example	What	is	the	fourth	highest	mountain	in	the	world	?
LOC	0.47	0.75	0.95	0.67	0.48	0.55	1.19	1.11	0.85	0.91

Figure 6. (Experiment III) Two examples in the question classification task. The upper row gives the sample sentence; the lower row shows the POPQORN (2-norm) lower bounds of individual words. "ENTY" and "LOC" represent *entity* and *location*, respectively.

(words), which guarantee robust classifier decisions as long as the perturbation magnitude in the word embedding space is smaller than the certificate. The 3 most sensitive words (words with 3 smallest bounds) are underlined. In the first example, the question is classified as "ENTY"(entity). Correspondingly, **name** is among the three most sensitive words, which is consistent with human cognition. In the second example, the question is classified as "LOC"(location). Again, **mountain** is shortlisted in the top three sensitive words. More examples are provided in the appendix Section B.4. Such observed consistency suggests POPQORN's potential in distinguishing the importance of different words.

5. Conclusion

This paper has proposed, for the first time, a robustness quantification framework called *POPQORN* that handles various RNN architectures including vanilla RNNs, LSTMs and GRUs. The certified bound gives a guaranteed lower bound of the minimum distortion in RNN adversaries, in contrast to the upper bound suggested by attacks. Experiments on different RNN tasks have demonstrated that POPQORN can compute non-trivial certified bounds, and provide insights on the importance and sensitivity of a single frame in sequence data.

Acknowledgment

The authors would like to thank Zhuolun Leon He for useful discussion. This work is partially supported by the Big Data Collaboration Research grant from SenseTime Group (CUHK Agreement No. TS1610626), the General Research Fund (Project 14236516, 17246416) of the Hong Kong Research Grants Council, MIT-IBM program, MIT-Skoltech program, and MIT-SenseTime program.

References

Boopathy, A., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, Jan 2019.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*, pp. 39–57, 2017.

Carlini, N. and Wagner, D. A. Audio adversarial examples: Targeted attacks on speech-to-text. *CoRR*, abs/1801.01944, 2018.

Cheng, C.-H., Nührenberg, G., and Ruess, H. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 251–268, 2017.

Cheng, M., Yi, J., Zhang, H., Chen, P., and Hsieh, C. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *CoRR*, abs/1803.01128, 2018.

Cissé, M., Adi, Y., Neverova, N., and Keshet, J. Houdini: Fooling deep structured prediction models. *CoRR*, abs/1707.05373, 2017.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., and Kohli, P. A dual approach to scalable verification of deep networks. *UAI*, 2018.

Ebrahimi, J., Lowd, D., and Dou, D. On adversarial examples for character-level neural machine translation. In *COLING*, pp. 653–663, 2018a.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. Hotflip: White-box adversarial examples for text classification. In *ACL*, pp. 31–36, 2018b.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pp. 1625–1634, 2018.

Gao, J., Lanchantin, J., Soffa, M. L., and Qi, Y. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops*, pp. 50–56, 2018.

Gong, Y. and Poellabauer, C. Crafting adversarial examples for speech paralinguistics applications. *CoRR*, abs/1711.03280, 2017.

Hein, M. and Andriushchenko, M. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2017.

Jia, R. and Liang, P. Adversarial examples for evaluating reading comprehension systems. In *EMNLP*, pp. 2021–2031, 2017.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, pp. 97–117, 2017.

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *ICLR Workshop*, 2017.

Li, X. and Roth, D. Learning question classifiers. In *COLING*, pp. 1–7, 2002.

Papernot, N., McDaniel, P., Swami, A., and Harang, R. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM*, pp. 49–54, 2016a.

Papernot, N., McDaniel, P. D., Swami, A., and Harang, R. E. Crafting adversarial input sequences for recurrent neural networks. *MILCOM*, pp. 49–54, 2016b.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *ICLR*, 2018.

Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1528–1540, 2016.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *NIPS*, pp. 10825–10836. 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *ICLR*, 2014.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. Towards fast computation of certified robustness for relu networks. *ICML*, 2018a.

Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., and Daniel, L. Evaluating the robustness of neural networks: An extreme value theory approach. In *ICLR*, 2018b.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *NIPS*, pp. 4944–4953. 2018.

Zhao, Z., Dua, D., and Singh, S. Generating natural adversarial examples. In *ICLR*, 2018.