RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and Its Applications

Huan Zhang*
UCLA
huan@huan-zhang.com

Pengchuan Zhang
Microsoft Research AI
penzhan@microsoft.com

Cho-Jui Hsieh
UCLA
chohsieh@cs.ucla.edu

Abstract

The Jacobian matrix (or the gradient for single-output networks) is directly related to many important properties of neural networks, such as the function landscape, stationary points, (local) Lipschitz constants and robustness to adversarial attacks. In this paper, we propose a *recursive* algorithm, **RecurJac**, to compute both upper and lower bounds for each element in the Jacobian matrix of a neural network with respect to network's input, and the network can contain a wide range of activation functions. As a byproduct, we can efficiently obtain a (local) Lipschitz constant, which plays a crucial role in neural network robustness verification, as well as the training stability of GANs. Experiments show that (local) Lipschitz constants produced by our method is of better quality than previous approaches, thus providing better robustness verification results. Our algorithm has polynomial time complexity, and its computation time is reasonable even for relatively large networks. Additionally, we use our bounds of Jacobian matrix to characterize the landscape of the neural network, for example, to determine whether there exist stationary points in a local neighborhood.

Introduction

Deep neural networks have been successfully applied to many applications, but one of the major criticisms is their being black boxes—no satisfactory explanation of their behavior can be easily offered. Given a neural network $f(\cdot)$ with input x, one fundamental question to ask is: how does a perturbation in the input space affect the output prediction? To formally answer this question and bound the behavior of neural networks, a critical step to answer this question is to compute the uniform bounds of the Jacobian matrix $\frac{\partial f(x)}{\partial x}$ for all x within a certain region. Many recent works on understanding or verifying the behavior of neural networks rely on this quantity. For example, once a (local) Jacobian bound is computed, one can immediately know the radius of a guaranteed "safe region" in the input space, where no adversarial perturbation can change the output label (Hein and Andriushchenko 2017; Weng et al. 2018b). This is also referred to as the *robustness verification problem*. In generative adversarial networks (GANs) (Goodfellow et al. 2014), the training process suffers from the gradient vanishing problem and can be very unstable. Adding the Lipschitz constant of the discriminator network as a constraint (Arjovsky, Chintala, and Bottou 2017; Miyato et al. 2018) or as a regularizer (Gulrajani et al. 2017) significantly improves the training stability of GANs. For neural networks, the Jacobian matrix $\frac{\partial f(x)}{\partial x}$ is also closely related to its Jacobian matrix with respect to the weights $\frac{\partial f(x;W)}{\partial W}$, whose bound directly characterizes the generalization gap in supervised learning and GANs; see, e.g., (Vapnik and Vapnik 1998; Sriperumbudur et al. 2009; Bartlett, Foster, and Telgarsky 2017; Arora and Zhang 2018; Zhang et al. 2018b).

In this paper, we propose a novel *recursive* algorithm, dubbed *RecurJac*, for efficiently computing a certified *Jacobian* bound. Unlike the layer-by-layer algorithm (Fast-Lip) for ReLU network in (Weng et al. 2018b), we develop a recursive refinement procedure that significantly outperforms Fast-Lip on ReLU networks, and our algorithm is general enough to be applied to networks with most common activation functions, not limited to ReLU. Our key observation is that the Jacobian bounds of previous layers can be used to reduce the uncertainties of neuron activations in the current layer, and some uncertain neurons can be fixed without affecting the final bound. We can then absorb these fixed neurons into the previous layers' weight matrix, which results in bounding Jacobian matrix for another shallower network. This technique can be applied recursively to get a tighter final bound. Compared with the non-recursive algorithm (Fast-Lip), RecurJac increases the computation cost by at most H times (H is depth of the network), which is reasonable even for relatively large networks.

We apply RecurJac to various applications. First, we can investigate the local optimization landscape after obtaining the upper and lower bounds of Jacobian matrix, by guaranteeing that no stationary points exist inside a certain region. Experimental results show that the radius of this region steadily decreases when networks become deeper. Second, RecurJac can find a local Lipschitz constant, which up to two magnitudes smaller than the state-of-the-art algorithm without a recursive structure (Figure 1). Finally, we can use RecurJac to evaluate the robustness of neural networks, by giving a certified lower bound within which no adversarial examples can be found.

*Work done during internship at Microsoft Research

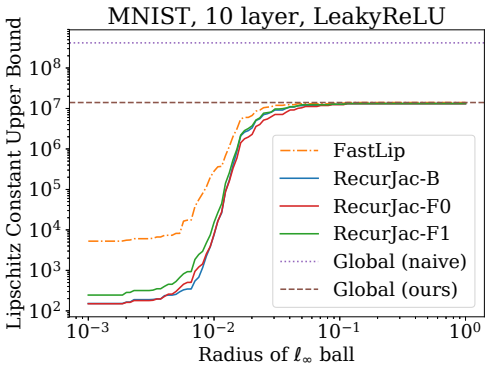


Figure 1: RecurJac can obtain local and global Lipschitz constants magnitudes better than existing algorithms.

Related Work

Computing Lipschitz constant. Computing a local or global Lipschitz constant of neural networks is a special case of our problem. One simple approach for estimating the Lipschitz constant for any black-box function is to sample many x, y and compute the maximal $\|f(x)-f(y)\|/\|x-y\|$ (Wood and Zhang 1996). However, the computed value may be an under-estimation unless the sample size goes to infinity. The Extreme Value Theory (De Haan and Ferreira 2007) can be used to refine the bound but the computed value could still under-estimate the Lipschitz constant (Weng et al. 2018b), especially due to the high dimensionality of inputs.

For a neural network with known structure and weights, it is possible to compute Lipschitz constant explicitly. An easy way to obtain a loose global Lipschitz constant is to multiply weight matrices' operator norms and the maximum derivative of each activation function. Since this quantity is simple to compute and can be optimized by back-propagation, many recent works also propose defenses to adversarial examples (Cisse et al. 2017; Elsayed et al. 2018; Tsuzuku, Sato, and Sugiyama 2018; Qian and Wegman 2018) or techniques to improve the training stability of GAN (Miyato et al. 2018) by regularizing this global Lipschitz constant. However, it is clearly a very loose Lipschitz constant, as will be shown in our experiments.

For 2-layer ReLU networks, (Raghunathan, Steinhardt, and Liang 2018) computes a global Lipschitz constant by relaxing the problem to semi-definite programming (SDP) and solving its dual, but it is computationally expensive. For 2-layer networks with twice differentiable activation functions, (Hein and Andriushchenko 2017) derives the local Lipschitz constant for robustness verification. These methods show promising results for 2-layer networks, but cannot be trivially extended to networks with multiple layers.

Bounds for Jacobian matrix. Recently, (Weng et al. 2018a) proposes an layer-by-layer algorithm, Fast-Lip, for computing the lower and upper bounds of Jacobian matrix with respect to network input x. It exploits the special activation patterns in ReLU networks but does not apply to networks with general activation functions. Most importantly, it loses power quickly when the network becomes deeper. Using Fast-Lip for robustness verification produces non-trivial bounds only for very shallow networks (less than 4 layers).

Robustness verification of neural networks. Assuming the output of a multi-class classification network $f(x)$ is a K-dimensional vector where each $f_j(x)$ is the logit for the j-th class and the final prediction $F(x) = \arg\max_j f_j(x)$, the following lemma gives a robustness lower bound (Hein and Andriushchenko 2017; Weng et al. 2018b):

Lemma 1. *For an input example x,*

$$F(x+\Delta) = y \text{ for all } \|\Delta\| < \min\Big\{R, \min_{j\neq y} \frac{f_y(x) - f_j(x)}{L_j}\Big\}, \tag{1}$$

where L_j is the Lipschitz constant of $f_j(x) - f_y(x)$ in some local region (will be formally defined later).

Therefore, as long as a local Lipschitz constant can be computed, we can verify that the prediction of a neural network will stay unchanged for any perturbation within radius R. A good local Lipschitz constant is hard to compute in general: (Hein and Andriushchenko 2017) only shows the results for 2-layer neural networks; (Weng et al. 2018b) applies a sampling-based approach and cannot guarantee that the computed radius satisfies (1). Thus, an efficient, guaranteed and tight bound for Lipschitz constant is essential for understanding the robustness of deep neural networks.

Other methods have also been proposed for robustness verification, including direct linear bounds (Zhang et al. 2018a; Croce, Andriushchenko, and Hein 2018; Weng et al. 2018a), convex adversarial polytope (Wong and Kolter 2018; Wong et al. 2018), Lagrangian relaxation (Dvijotham et al. 2018) and geometry abstraction (Gehr et al. 2018; Mirman, Gehr, and Vechev 2018). In this paper we focus on Local Lipschitz constant based methods only.

RecurJac: Recursive Jacobian Bounding

In this section, we present RecurJac, our recursive algorithm for uniformly bounding (local) Jacobian matrix of neural networks with a wide range of activation functions.

Notations. For an H-layer neural network $f(x)$ with input $x \in \mathbb{R}^{n_0}$, weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and bias vectors $b^{(l)} \in \mathbb{R}^{n_l}$, the network $f(x)$ can be defined recursively as $h^{(l)}(x) = \sigma^{(l)}(\mathbf{W}^{(l)}h^{(l-1)}(x)+b^{(l)})$ for all $l \in \{1, \ldots, H-1\}$ with $h^{(0)} := x$, $f(x) = \mathbf{W}^{(H)}h^{(H-1)}(x) + b^{(H)}$. $\sigma^{(l)}$ is a component-wise activation function of (leaky-)ReLU, sigmoid family (including sigmoid, arctan, hyperbolic tangent, etc), and other activation functions that satisfy the assumptions we will formally show below. We denote $\mathbf{W}^{(l)}_{r,:}$ as the r-th row and $\mathbf{W}^{(l)}_{:,j}$ as the j-th column of $\mathbf{W}^{(l)}$. For convenience, we denote $f^{(l)}(x) := \mathbf{W}^{(l)}h^{(l-1)}(x) + b^{(l)}$ as the pre-activation function values.

Local Lipchitz constant. Given a function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ and two distance metrics d and d' on $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively, the local Lipschitz constant $L^S_{d,d'}$ of f in a close

ball of radius R centered at s (denoted as $S = B_d[s; R]$) is defined as:

$$d'(f(x), f(y)) \leqslant L^S_{d,d'} d(x, y), \text{for all } x, y \in S := B_d[s; R]$$

Any scalar $L^S_{d,d'}$ that satisfies this condition is a local Lipschitz constant. A good local Lipschitz constant should be as small as possible, *i.e.*, close to *the best* (smallest) local Lipschitz constant. A Lipschitz constant we compute can be seen as an upper bound of the best Lipschitz constant.

Assumptions on activation functions. RecurJac has the following assumptions on the activation function $\sigma(x)$:
Assumption 1. $\sigma(x)$ is continuous and differentiable almost everywhere on $\mathbb{R}$. This is a basic assumption for neural network activation functions.
Assumption 2. There exists a positive constant C such that $0 \leqslant \sigma'(x) \leqslant C$ when the derivative exists. This covers all common activation functions, including (leaky-)ReLU, hard-sigmoid, exponential linear units (ELU), sigmoid, tanh, arctan and all sigmoid-shaped family activation functions. This assumption helps us derive an elegant bound.

Overview of Techniques. The local Lipschitz constant can be presented as the maximum directional derivative inside the ball $B_d[s; R]$ (Weng et al. 2018b). For differentiable functions, this is the maximum norm of gradient with respect to the distance metric d (or the maximal operator norm of Jacobian induced by distances d' and d in the vector-output case). We bound each element of Jacobian through a layer-by-layer approach, as shown below.

Define diagonal matrices Σ representing the derivatives of the activation functions:

$$\Sigma^{(l)} := \text{diag}\{\sigma'(f^{(l)}(x))\}.$$

The Jacobian matrix of a H-layer network can be written as:

$$\nabla f^{(H)}(x) = \mathbf{W}^{(H)}\Sigma^{(H-1)}\mathbf{W}^{(H-1)} \dots \mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}. \tag{2}$$

For the ease of notation, we also define

$$\mathbf{Y}^{(-l)} := \frac{\partial f^{(H)}}{\partial h^{(l-1)}} = \mathbf{W}^{(H)}\Sigma^{(H-1)} \dots \mathbf{W}^{(l+1)}\Sigma^{(l)}\mathbf{W}^{(l)}$$

for $l \in [H]$. As a special case, $\mathbf{Y}^{(-1)} := \nabla f^{(H)}$.

In the first step, we assume that we have the following pre-activation bounds $\boldsymbol{l}^{(l)}_r$ and $\boldsymbol{u}^{(l)}_r$ for every layer $l \in [H-1]$:

$$\boldsymbol{l}^{(l)}_r \leqslant f^{(l)}_r(x) \leqslant \boldsymbol{u}^{(l)}_r \quad \forall r \in [n_l], x \in B_d[s; R] \tag{3}$$

We can get these bounds efficiently via any algorithms that compute layer-wise activation bounds, including CROWN (Zhang et al. 2018a) and convex adversarial polytope (Wong and Kolter 2018). Because pre-activations are within some ranges rather than fixed values, Σ matrices contain uncertainties, which will be characterized analytically.

In the second step, we compute both lower and upper bounds for each entry of $\mathbf{Y}^{(-l)} := \frac{\partial f^{(H)}}{\partial h^{(l-1)}}$ in a backward manner. More specifically, we compute $\mathbf{L}^{(-l)}, \mathbf{U}^{(-l)} \in \mathbb{R}^{n_H \times n_{l-1}}$ so that

$$\mathbf{L}^{(-l)} \leqslant \mathbf{Y}^{(-l)}(x) \leqslant \mathbf{U}^{(-l)} \quad \forall x \in B_d[s; R] \tag{4}$$

holds true element-wisely. For layer H, we have $\mathbf{Y}^{(-H)} = \mathbf{W}^{(H)}$ and thus $\mathbf{L}^{(-H)} = \mathbf{U}^{(-H)} = \mathbf{W}^{(H)}$. For layers $l < H$, uncertainties in Σ matrices propagate into $\mathbf{Y}^{(-l)}$ layer by layer. Naively deriving $(\mathbf{L}^{(-l+1)}, \mathbf{U}^{(-l+1)})$ just from $(\mathbf{L}^{(-l)}, \mathbf{U}^{(-l)})$ and $(\boldsymbol{l}^{(l-1)}, \boldsymbol{u}^{(l-1)})$ leads to a very loose bound. We propose a fast recursive algorithm that makes use of bounds for all previous layers to compute a much tighter bound for $\mathbf{Y}^{(-l)}$. Applying our algorithm to $\mathbf{Y}^{(-H+1)}, \mathbf{Y}^{(-H+2)}, \cdots$ will eventually allow us to obtain $\mathbf{Y}^{(-1)}$. Our algorithm can also be applied in a forward manner; the forward version (RecurJac-F) typically leads to slightly tighter bounds but can slow down the computation significantly, as we will show in the experiments.

From an optimization perspective, we essentially try to solve two constrained maximization and minimization problems with variables $\Sigma^{(l)}_{r,r}$, for each element $\{j, k\}$ in the Jacobian $\nabla f^{(H)}(x)$:

$$\max_{\boldsymbol{l}^{(l)}_r \leqslant \Sigma^{(l)}_{r,r} \leqslant \boldsymbol{u}^{(l)}_r} [\nabla f^{(H)}(x)]_{j,k} \quad \text{and} \quad \min_{\boldsymbol{l}^{(l)}_r \leqslant \Sigma^{(l)}_{r,r} \leqslant \boldsymbol{u}^{(l)}_r} [\nabla f^{(H)}(x)]_{j,k}. \tag{5}$$

Raghunathan, Steinhardt, and Liang (2018) show that even for ReLU networks with one hidden layer, finding the maximum ℓ_1 norm of the gradient is equivalent to the Max-Cut problem and NP-complete. RecurJac is a polynomial time algorithm to give upper and lower bounds on $[\nabla f^{(H)}(x)]_{j,k}$, rather than solving the exact maxima and minima in exponential time.

After obtaining the Jacobian bounds $\mathbf{Y}^{(-1)} := \nabla f^{(H)}$, we can make use of it to derive an upper bound for the local Lipchitiz constant in the set $S = B_d[s; R]$. We present bounds when d and d' are both ordinary p-norm ($p = [1, +\infty) \cup \{+\infty\}$) distance in Euclidean space. We can also use the Jacobian bounds for other proposes, like understanding the local optimization landscape.

Bounds for $\Sigma^{(l)}$

From (2), we can see that the uncertainties in $\nabla f^{(H)}$ are purely from $\sigma'(f^{(l)}(x))$; all $\mathbf{W}^{(l)}$ are fixed. For any $l \in [H-1]$, we define the range of $\sigma'(f^{(l)}_r(x))$ as $\boldsymbol{l}'^{(l)}_r$ and $\boldsymbol{u}'^{(l)}_r$, i.e.,

$$\boldsymbol{l}'^{(l)}_r \leqslant \sigma'(f^{(l)}_r(x)) \leqslant \boldsymbol{u}'^{(l)}_r \quad \forall r \in [n_l]. \tag{6}$$

Note that $\boldsymbol{l}'^{(l)}_r$ and $\boldsymbol{u}'^{(l)}_r$ can be easily obtained because we know $\boldsymbol{l}^{(l)}_r \leqslant f^{(l)}_r(x) \leqslant \boldsymbol{u}^{(l)}_r$ (thanks to (3)) and the analytical form of $\sigma'(x)$. For example, for the sigmoid function $\sigma(x) = \frac{e^x}{1+e^x}$, $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, we have:

$$\boldsymbol{l}'^{(l)}_r = \begin{cases} \sigma'(\boldsymbol{l}^{(l)}_r) & \text{if } \boldsymbol{l}^{(l)}_r \leqslant \boldsymbol{u}^{(l)}_r \leqslant 0; \\ \sigma'(\boldsymbol{u}^{(l)}_r) & \text{if } \boldsymbol{u}^{(l)}_r \geqslant \boldsymbol{l}^{(l)}_r \geqslant 0; \\ \sigma'(\max\{-\boldsymbol{l}^{(l)}_r, \boldsymbol{u}^{(l)}_r\}) & \text{if } \boldsymbol{l}^{(l)}_r \leqslant 0 \leqslant \boldsymbol{u}^{(l)}_r. \end{cases} \tag{7}$$

$$\boldsymbol{u}'^{(l)}_r = \begin{cases} \sigma'(\boldsymbol{u}^{(l)}_r) & \text{if } \boldsymbol{l}^{(l)}_r \leqslant \boldsymbol{u}^{(l)}_r \leqslant 0; \\ \sigma'(\boldsymbol{l}^{(l)}_r) & \text{if } \boldsymbol{u}^{(l)}_r \geqslant \boldsymbol{l}^{(l)}_r \geqslant 0; \\ \sigma'(0) & \text{if } \boldsymbol{l}^{(l)}_r \leqslant 0 \leqslant \boldsymbol{u}^{(l)}_r. \end{cases} \tag{8}$$

Equations (7) and (8) are also valid for other sigmoid-family activation functions, including $\sigma(x) = \frac{e^x}{1+e^x}$, $\sigma(x) = \tanh(x)$, $\sigma(x) = \arctan(x)$ and many others.

For (leaky-)ReLU activation functions with a negative-side slope α ($0 \leqslant \alpha \leqslant 1$), $\boldsymbol{l}_r'^{(l)}$ and $\boldsymbol{u}_r'^{(l)}$ are:

$$\boldsymbol{l}_r'^{(l)} = \begin{cases} \alpha & \text{if } \boldsymbol{l}_r^{(l)} \leqslant \boldsymbol{u}_r^{(l)} \leqslant 0 \quad \text{or} \quad \boldsymbol{l}_r^{(l)} \leqslant 0 \leqslant \boldsymbol{u}_r^{(l)}; \\ 1 & \text{if } \boldsymbol{u}_r^{(l)} \geqslant \boldsymbol{l}_r^{(l)} \geqslant 0. \end{cases}$$

$$\boldsymbol{u}_r'^{(l)} = \begin{cases} \alpha & \text{if } \boldsymbol{l}_r^{(l)} \leqslant \boldsymbol{u}_r^{(l)} \leqslant 0; \\ 1 & \text{if } \boldsymbol{u}_r^{(l)} \geqslant \boldsymbol{l}_r^{(l)} \geqslant 0 \quad \text{or} \quad \boldsymbol{l}_r^{(l)} \leqslant 0 \leqslant \boldsymbol{u}_r^{(l)}. \end{cases}$$

For (leaky-)ReLU activation functions, in the cases where $\boldsymbol{l}_r^{(l)} \leqslant \boldsymbol{u}_r^{(l)} \leqslant 0$ and $\boldsymbol{u}_r^{(l)} \geqslant \boldsymbol{l}_r^{(l)} \geqslant 0$, we have $\boldsymbol{l}_r'^{(l)} = \boldsymbol{u}_r'^{(l)}$, so $\Sigma_{r,r}^{(l)}$ becomes a constant and there is no uncertainty.

A recursive algorithm to bound $\mathbf{Y}^{(-l)}$

Bounds for $\mathbf{Y}^{(-H+1)}$. By definition, we have $\mathbf{Y}^{(-H)} = \mathbf{W}^{(H)}$ and $\mathbf{Y}^{(-H+1)} = \mathbf{Y}^{(-H)}\Sigma^{(H-1)}\mathbf{W}^{(H-1)}$. Thus,

$$\mathbf{Y}_{j,k}^{(-H+1)} = \sum_{r \in [n_{H-1}]} \mathbf{W}_{j,r}^{(H)} \sigma'(f_r^{(H-1)}) \mathbf{W}_{r,k}^{(H-1)}, \quad (9)$$

where $\boldsymbol{l}_r'^{(H-1)} \leqslant \sigma'(f_r^{(H-1)}) \leqslant \boldsymbol{u}_r'^{(H-1)}$ thanks to (6).

By assumption 2, $\sigma'(x)$ is always non-negative, and thus we only need to consider the signs of $\mathbf{W}^{(H)}$ and $\mathbf{W}^{(H-1)}$. Denote $\mathbf{L}_{j,k}^{(-H+1)}$ and $\mathbf{U}_{j,k}^{(-H+1)}$ to be a lower and upper bounds of (9). By examining the signs of each term, we have

$$\begin{aligned} \mathbf{L}_{j,k}^{(-H+1)} = & \sum_{\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)} < 0} \boldsymbol{u}_r'^{(H-1)} \mathbf{W}_{j,r}^{(H)} \mathbf{W}_{r,k}^{(H-1)} \\ & + \sum_{\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)} > 0} \boldsymbol{l}_r'^{(H-1)} \mathbf{W}_{j,r}^{(H)} \mathbf{W}_{r,k}^{(H-1)}, \end{aligned} \quad (10)$$

$$\begin{aligned} \mathbf{U}_{j,k}^{(-H+1)} = & \sum_{\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)} > 0} \boldsymbol{u}_r'^{(H-1)} \mathbf{W}_{j,r}^{(H)} \mathbf{W}_{r,k}^{(H-1)} \\ & + \sum_{\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)} < 0} \boldsymbol{l}_r'^{(H-1)} \mathbf{W}_{j,r}^{(H)} \mathbf{W}_{r,k}^{(H-1)}. \end{aligned} \quad (11)$$

In (10), we collect all negative terms of $\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)}$ and multiply them by $\boldsymbol{u}_r'^{(H-1)}$ as a lower bound of $\sum_{\mathbf{W}_{j,r}^{(H)}\mathbf{W}_{r,k}^{(H-1)} < 0} \sigma'(f_r^{(H-1)}(x)) \mathbf{W}_{j,r}^{(H)} \mathbf{W}_{r,k}^{(H-1)}$, and collect all positive terms and multiply them by $\boldsymbol{l}_r'^{(H-1)}$ as a lower bound of the positive counterpart. We obtain the upper bound in (11) following the same rationale. Fast-Lip is a special case of RecurJac when there are only two layers with ReLU activations; RecurJac becomes much more sophisticated in multi-layer cases, as we will show below.

Bounds for $\mathbf{Y}^{(-l)}$ when $1 \leqslant l < H-1$. By definition, we have $\mathbf{Y}^{(-l+1)} = \mathbf{Y}^{(-l)}\Sigma^{(l-1)}\mathbf{W}^{(l-1)}$, i.e.,

$$\mathbf{Y}_{j,k}^{(-l+1)} = \sum_{r \in [n_{l-1}]} \mathbf{Y}_{j,r}^{(-l)} \sigma'(f_r^{(l-1)}(x)) \mathbf{W}_{r,k}^{(l-1)}, \quad (12)$$

where $\boldsymbol{l}_r'^{(l-1)} \leqslant \sigma'(f_r^{(l-1)}(x)) \leqslant \boldsymbol{u}_r'^{(l-1)}$ thanks to (6) and

$$\mathbf{L}_{j,r}^{(-l)} \leqslant \mathbf{Y}_{j,r}^{(-l)} \leqslant \mathbf{U}_{j,r}^{(-l)} \quad \forall j, r$$

thanks to previous computation. We want to find the bounds

$$\mathbf{L}_{j,k}^{(-l+1)} \leqslant \mathbf{Y}_{j,k}^{(-l+1)} \leqslant \mathbf{U}_{j,k}^{(-l+1)} \quad \forall j, k.$$

We decompose (12) into two terms:

$$\begin{aligned} \mathbf{Y}_{j,k}^{(-l+1)} = & \underbrace{\sum_{\{r\,:\,\mathbf{L}_{j,r}^{(-l)} < 0 < \mathbf{U}_{j,r}^{(-l)}\}} \mathbf{Y}_{j,r}^{(-l)} \sigma'(f_r^{(l-1)}(x)) \mathbf{W}_{r,k}^{(l-1)}}_{I} \\ & + \underbrace{\sum_{\{r\,:\,\mathbf{L}_{j,r}^{(-l)} \geqslant 0 \text{ or } \mathbf{U}_{j,r}^{(-l)} \leqslant 0\}} \mathbf{Y}_{j,r}^{(-l)} \sigma'(f_r^{(l-1)}(x)) \mathbf{W}_{r,k}^{(l-1)}}_{II}, \end{aligned} \quad (13)$$

and bound them separately.

Observing the signs of each term in I and $\boldsymbol{u}_r'^{(l+1)} \geqslant \boldsymbol{l}_r'^{(l+1)} \geqslant 0$, we take:

$$\mathbf{L}_{j,k}^{(-l+1),\pm} = \sum_{\mathbf{W}_{r,k}^{(l-1)} < 0} \boldsymbol{u}_r'^{(l-1)} \mathbf{U}_{j,r}^{(-l)} \mathbf{W}_{r,k}^{(l-1)} + \sum_{\mathbf{W}_{r,k}^{(l-1)} > 0} \boldsymbol{u}_r'^{(l-1)} \mathbf{L}_{j,r}^{(-l)} \mathbf{W}_{r,k}^{(l-1)} \quad (14)$$

$$\mathbf{U}_{j,k}^{(-l+1),\pm} = \sum_{\mathbf{W}_{r,k}^{(l-1)} < 0} \boldsymbol{u}_r'^{(l-1)} \mathbf{L}_{j,r}^{(-l)} \mathbf{W}_{r,k}^{(l-1)} + \sum_{\mathbf{W}_{r,k}^{(l-1)} > 0} \boldsymbol{u}_r'^{(l-1)} \mathbf{U}_{j,r}^{(-l)} \mathbf{W}_{r,k}^{(l-1)} \quad (15)$$

The index constraint $\{r : \mathbf{L}_{j,r}^{(-l)} < 0 < \mathbf{U}_{j,r}^{(-l)}\}$ is still effective in (14) and (15), but we omit it for notation simplicity. Then we can show that $\mathbf{L}_{j,k}^{(-l+1),\pm}$ and $\mathbf{U}_{j,k}^{(l+1),\pm}$ are a lower and upper bound for term I in (13) as follows.

Proposition 1.

$$\mathbf{L}_{j,k}^{(-l+1),\pm} \leqslant I \leqslant \mathbf{U}_{j,k}^{(-l+1),\pm}, \quad (16)$$

where I is the first term in (13)*.*

For term II in (13), the sign of $\mathbf{Y}_{j,r}^{(-l)}$ does not change since $\mathbf{L}_{j,r}^{(-l)} \geqslant 0$ or $\mathbf{U}_{j,r}^{(-l)} \leqslant 0$. Similar to what we did in (10) and (11), depending on the sign of $\mathbf{Y}_{j,r}^{(-l)}\mathbf{W}_{r,k}^{(l-1)}$, we can lower/upper bound term II using $\mathbf{Y}^{(-l)}$ itself instead of its bound $(\mathbf{L}^{(-l)}, \mathbf{U}^{(-l)})$. This will give us much tighter bounds than just naively using $(\mathbf{L}^{(-l)}, \mathbf{U}^{(-l)})$ as we deal with term I. More specifically, we define $2n_H$ matrices $\widetilde{\mathbf{W}}^{(l,l-1,j)}, \widehat{\mathbf{W}}^{(l,l-1,j)} \in \mathbb{R}^{n_l \times n_{l-2}}$ for $j \in [n_H]$ as below:

$$\begin{aligned} \widetilde{\mathbf{W}}_{i,k}^{(l,l-1,j)} = & \sum_{\substack{\mathbf{L}_{j,r}^{(-l)} \geqslant 0, \mathbf{W}_{r,k}^{(l-1)} > 0 \\ \text{or } \mathbf{U}_{j,r}^{(-l)} \leqslant 0, \mathbf{W}_{r,k}^{(l-1)} < 0}} \mathbf{W}_{i,r}^{(l)} \boldsymbol{l}_r'^{(l-1)} \mathbf{W}_{r,k}^{(l-1)} \\ & + \sum_{\substack{\mathbf{L}_{j,r}^{(-l)} \geqslant 0, \mathbf{W}_{r,k}^{(l-1)} < 0 \\ \text{or } \mathbf{U}_{j,r}^{(-l)} \leqslant 0, \mathbf{W}_{r,k}^{(l-1)} > 0}} \mathbf{W}_{i,r}^{(l)} \boldsymbol{u}_r'^{(l-1)} \mathbf{W}_{r,k}^{(l-1)}, \end{aligned} \quad (17)$$

$$\widehat{\mathbf{W}}_{i,k}^{(l,l-1,j)} = \sum_{\substack{\mathbf{L}_{j,r}^{(-l)}\geqslant 0, \mathbf{W}_{r,k}^{(l-1)}>0 \\ \text{or } \mathbf{U}_{j,r}^{(-l)}\leqslant 0, \mathbf{W}_{r,k}^{(l-1)}<0}} \mathbf{W}_{i,r}^{(l)} \boldsymbol{u}_r'^{(l-1)} \mathbf{W}_{r,k}^{(l-1)} + \sum_{\substack{\mathbf{L}_{j,r}^{(-l)}\geqslant 0, \mathbf{W}_{r,k}^{(l-1)}<0 \\ \text{or } \mathbf{U}_{j,r}^{(-l)}\leqslant 0, \mathbf{W}_{r,k}^{(l-1)}>0}} \mathbf{W}_{i,r}^{(l)} \boldsymbol{l}_r'^{(l-1)} \mathbf{W}_{r,k}^{(l-1)}. \tag{18}$$

Then we can show the following lemma.

Lemma 2. *For any $j \in [n_H]$, we have*

$$\mathbf{Y}_{j,:}^{(-l-1)} \Sigma^{(l)} \widecheck{\mathbf{W}}_{:,k}^{(l,l-1,j)} \leqslant II \leqslant \mathbf{Y}_{j,:}^{(-l-1)} \Sigma^{(l)} \widehat{\mathbf{W}}_{:,k}^{(l,l-1,j)}, \tag{19}$$

where II is the second term in (13)*.*

Note that when the sign of $\mathbf{Y}_{j,r}^{(-l)}$ is fixed, i.e., $\mathbf{L}_{j,r}^{(-l)} \geqslant 0$ or $\mathbf{U}_{j,r}^{(-l)} \leqslant 0$ in term II, the bounds in (19) is always tighter than those in (16). After we know the sign of $\mathbf{Y}_{j,r}^{(-l)}$, we can fix $\sigma'(f_r^{(l-1)}(x))$ to be either $\boldsymbol{l}_r'^{(l-1)}$ or $\boldsymbol{u}_r'^{(l-1)}$ according to the sign of $\mathbf{W}_{r,k}^{(l-1)}$ and thus eliminate the uncertainty in $\sigma'(f_r^{(l-1)}(x))$. Then we can plug $\mathbf{Y}_{j,r}^{(-l)} = \sum_s \mathbf{Y}_{j,i}^{(-l-1)} \sigma'(f_i^{(l)}(x)) \mathbf{W}_{i,r}^{(l)}$ into the lower and upper bounds and merge terms involving $\mathbf{W}_{i,r}^{(l)}$, $\sigma'(f_r^{(l-1)}(x))$ and $\mathbf{W}_{r,k}^{(l-1)}$, resulting in (19). Compared with using the worst-case bound $\mathbf{L}_{j,r}^{(-l)} \leqslant \mathbf{Y}_{j,r}^{(-l)} \leqslant \mathbf{U}_{j,r}^{(-l)}$ directly in (16), we expand $\mathbf{Y}_{j,r}^{(-l)}$ and remove uncertainty in $\sigma'(f_r^{(l-1)}(x))$ in (19), and thus get much tighter bounds.

Finally, combining Proposition 1 and Lemma 2, we get the following recursive formula to bound $\mathbf{Y}^{(-l+1)}$.

Theorem 1. *For any $1 < l < H$ and any $j \in [n_H]$, we have*

$$\mathbf{Y}_{j,:}^{(-l+1)} \geqslant \mathbf{L}_{j,:}^{(-l+1),\pm} + \mathbf{Y}_{j,:}^{(-l-1)} \Sigma^{(l)} \widecheck{\mathbf{W}}_{:,k}^{(l,l-1,j)}$$

and

$$\mathbf{Y}_{j,:}^{(-l+1)} \leqslant \mathbf{U}_{j,:}^{(-l+1),\pm} + \mathbf{Y}_{j,:}^{(-l-1)} \Sigma^{(l)} \widehat{\mathbf{W}}_{:,k}^{(l,l-1,j)},$$

where $\mathbf{L}^{(-l+1),\pm}$, $\mathbf{U}^{(-l+1),\pm}$, $\widecheck{\mathbf{W}}^{(l,l-1,j)}$ and $\widehat{\mathbf{W}}^{(l,l-1,j)}$ are defined in (14)*,* (15)*,* (17) *and* (18)*, respectively.*

Remark 1. The lower and upper bounds of $\mathbf{Y}^{(-H+1)}$ in (10) and (11) can be viewed as a special case of Theorem 1 when $l = H$. Because we have $\mathbf{L}^{(-H)} = \mathbf{U}^{(-H)} = \mathbf{W}^{(l)}$ in this case, we do not have term I in the decomposition (13). Moreover, the bounds of term II in (19) are reduced to exactly (10) and (11) after we notice that $\widecheck{\mathbf{W}}_{j,k}^{(H,H-1,j)} = \mathbf{L}_{j,k}^{(-H+1)}$ and $\widehat{\mathbf{W}}_{j,k}^{(H,H-1,j)} = \mathbf{U}_{j,k}^{(-H+1)}$ and specify $\mathbf{Y}^{(-H-1)} = \Sigma^{(H)} = I_{n_H}$. Specifying $\mathbf{Y}^{(-H-1)} = \Sigma^{(H)} = I_{n_H}$ is equivalent to adding another identity layer to the neural network $f^{(H)}(x)$.

A recursive algorithm to bound $\mathbf{Y}^{(-l)}$. Notice that the lower and upper bounds in Lemma 2 have exactly the same formation of $\mathbf{Y}^{(-l)} = \mathbf{Y}^{(-l-1)} \Sigma^{(l)} \mathbf{W}^{(l)}$, by replacing $\mathbf{W}^{(l)}$ with $\widecheck{\mathbf{W}}^{(l,l-1,j)}$ and $\widehat{\mathbf{W}}^{(l,l-1,j)}$. Therefore, we can recursively apply our Theorem 1 to obtain an lower and upper bound for $\mathbf{Y}^{(-l+1)}$, denoted as $\mathbf{L}^{(-l+1)}$ and $\mathbf{U}^{(-l+1)}$ separately. This recursive procedure further reduces uncertainty in Σ for all previous layers, improving the quality of bounds significantly. We elaborate our recursive algorithm in Algorithm 1 for the case $n_H = 1$, so we omit the last superscript $j = 1$ in $\widecheck{\mathbf{W}}^{(l,l-1,1)}$ and $\widehat{\mathbf{W}}^{(l,l-1,1)}$. When $n_H > 1$, we can apply Algorithm 1 independently for each output.

Algorithm 1 ComputeLU (compute the lower and upper Jacobian bounds)

Require: $\mathbf{W}^{(l)}$, bounds $\{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=l+1}^{H}$, $\{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=l+1}^{H}$

if $l = H$ **then**
  $\mathbf{L}^{(-l)} = \mathbf{U}^{(-l)} = \mathbf{W}^{(l)}$
else if $l = H - 1$ **then**
  Compute $\mathbf{L}^{(-l)}$ from (10), $\mathbf{U}^{(-l)}$ from (11)
else if $1 \leqslant l < H - 1$ **then**
  Compute $\widecheck{\mathbf{W}}^{(l+1,l)}$ from (17), $\widehat{\mathbf{W}}^{(l+1,l)}$ from (18)
  $(\mathbf{L}^{(-l-1,-l)}, \backsim)$ = ComputeLU($\widecheck{\mathbf{W}}^{(l+1,l)}$, $\{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=l+2}^{H}$, $\{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=l+2}^{H}$)
  $(\backsim, \mathbf{U}^{(-l-1,-l)})$ = ComputeLU($\widehat{\mathbf{W}}^{(l+1,l)}$, $\{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=l+2}^{H}$, $\{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=l+2}^{H}$)
  Compute $\mathbf{L}^{(-l),\pm}$ from (14), $\mathbf{U}^{(-l),\pm}$ from (15)
  $\mathbf{L}^{(-l)} = \mathbf{L}^{(-l),\pm} + \mathbf{L}^{(-l-1,-l)}$
  $\mathbf{U}^{(-l)} = \mathbf{U}^{(-l),\pm} + \mathbf{U}^{(-l-1,-l)}$
end if
return $\mathbf{L}^{(-l)}$, $\mathbf{U}^{(-l)}$

Compute the bounds in a forward manner. In previous sections, we start our computation from the last layer and bound $\mathbf{Y}^{(-l)} := \frac{\partial f^{(H)}}{\partial h^{(l-1)}}$ in a backward manner. By transposing (2), we have

$$[\nabla f^{(H)}(x)]^T = \mathbf{W}^{(1)T} \Sigma^{(1)} \mathbf{W}^{(2)T} \cdots \Sigma^{(H-1)} \mathbf{W}^{(H)T}.$$

Then we can apply Algorithm 1 to bound $\nabla f^{(H)}(x)^T$ according to the equation above. This is equivalent to starting from the first layer, and bound $\frac{\partial f^{(l)}}{\partial x}$ from $l = 1$ to H. Because we obtain the bounds of pre-activations in a forward manner by CROWN (Zhang et al. 2018a), the bounds (3) get looser when the layer index l gets larger. Therefore, bounding the Jacobian by the forward version is expected to get tighter bounds of $\frac{\partial f^{(l)}}{\partial x}$ at least for small l. In our experiments, we see that the bounds for $\nabla f^{(H)}(x)$ obtained from the forward version are typically a little tighter than those obtained from the backward version. However, the "output" dimension in this case is n_0, which is the input dimension of the neural network. For image classification networks, $n_H \ll n_0$, the forward version has to apply Algorithm 1 n_0 times to obtain the final bounds and thus increases the computational cost significantly compared to the backward version. We make a detailed comparison between the forward and backward version in the experiment section.

Compute a local Lipschitz constant

After obtaining $\mathbf{L}^{(-1)} \leqslant \mathbf{Y}^{(-1)} := \nabla f(x) \leqslant \mathbf{U}^{(-1)}$ for all $x \in S$, we define

$$\max_{x \in S} |[\nabla f(x)]| \leqslant \mathbf{M} := \max(|\mathbf{L}^{(-1)}|, |\mathbf{U}^{(-1)}|), \quad (20)$$

where the max and inequality are taken element-wise. In the rest of this subsection, we simplify the notations $\mathbf{Y}^{(-1)}, \mathbf{L}^{(-1)}, \mathbf{U}^{(-1)}$ to $\mathbf{Y}, \mathbf{L}, \mathbf{U}$ when no confusion arises.

Recall that the Local Lipschitz constant L_d^S can be evaluated as $L_{d,d'}^S = \max_{x \in S} \|\nabla f(x)\|_{d,d'}$. $\nabla f(x)$ is the Jacobian matrix and $\|\cdot\|_{d,d'}$ denotes the induced operator norm. Then we can bound the maximum norm of Jacobian (local Lipschitz constant) considering its element-wise worst case. When d, d' are both ordinary p-norm ($p = [1, +\infty) \cup \{+\infty\}$) distance in Euclidean space, we denote $L_{d,d'}$ as L_p, and it can be bounded as follows.

Proposition 2. *For any $1 \leqslant p \leqslant +\infty$, we have*

$$L_p^S := \max_{x \in B_{\ell_p}[s;R]} \|\nabla f(x)\|_p \leqslant \|\mathbf{M}\|_p, \quad (21)$$

where $\mathbf{M} := \max(|\mathbf{L}|, |\mathbf{U}|)$ is defined in (20).

Improve the bound for L_∞^S. For the important case of upper bounding L_∞^S, we use an additional trick to improve the bound (21). We note that $\|\nabla f(x)\|_\infty = \max_j \sum_k |\mathbf{Y}_{j,k}|$. As in (13), we decompose it into two terms

$$\sum_k |\mathbf{Y}_{j,k}| = \underbrace{\sum_{k \in \mathcal{T}_j} |\mathbf{Y}_{j,k}|}_{I} + \underbrace{\sum_{k \in \mathcal{T}_j^+} \mathbf{Y}_{j,k} - \sum_{k \in \mathcal{T}_j^-} \mathbf{Y}_{j,k}}_{II}, \quad (22)$$

where $\mathcal{T}_j^+ := \{k | \mathbf{L}_{j,k} \geqslant 0\}$, $\mathcal{T}_j^- := \{k | \mathbf{U}_{j,k} \leqslant 0\}$, and $\mathcal{T}_j := \{k | \mathbf{L}_{j,k} < 0 < \mathbf{U}_{j,k}\}$.

For term I, we take the same bound as we have in (21), i.e., $I \leqslant \sum_{k \in \mathcal{T}_j} \mathbf{M}_{j,k}$.

For term II, thanks to $\mathbf{Y} = \mathbf{Y}^{(-2)} \Sigma^{(1)} \mathbf{W}^{(1)}$, we have

$$II = \sum_r \mathbf{Y}_{j,r}^{(-2)} \sigma'(f_r^{(1)}(x)) \Big(\sum_{k \in \mathcal{T}_j^+} \mathbf{W}_{r,k}^{(1)} - \sum_{k \in \mathcal{T}_j^-} \mathbf{W}_{r,k}^{(1)} \Big).$$

Define $\widehat{w}^{(j)} \in \mathbb{R}^{n_1 \times 1}$ and

$$\widehat{w}_r^{(j)} := \sum_{k \in \mathcal{T}_j^+} \mathbf{W}_{r,k}^{(1)} - \sum_{k \in \mathcal{T}_j^-} \mathbf{W}_{r,k}^{(1)}. \quad (23)$$

Algorithm 2 Upper bound of $\max_{x \in B_{\ell_\infty}[s;R]} \|\nabla f(x)\|_\infty$

1: Compute $\mathbf{M}$ from (20)
2: **for** $j \in [n_H]$ **do**
3: Compute $\widehat{w}_r^{(j)}$ from (23)
4: $(\backsim, \mathbf{U}^{(0,j)})$ = ComputeLU($\widehat{w}_r^{(j)}$, $\{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=1}^H$, $\{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=1}^H$)
5: $\mathbf{U}_j^{(0)} = \mathbf{U}^{(0,j)} + \sum_{k \in \mathcal{T}_j} \mathbf{M}_{j,k}$
6: **end for**
7: **return** $\max_{j \in [n_H]} \mathbf{U}_j^{(0)}$

Combining upper bounds for both terms, we obtain

$$\sum_j |\mathbf{Y}_{i,j}| \leqslant \sum_{k \in \mathcal{T}_j} \mathbf{M}_{j,k} + \mathbf{Y}_{j,:}^{(-2)} \Sigma^{(1)} \widehat{w}^{(j)}$$

In the same flavor with Theorem 1, this bound avoids the worst case bound $\mathbf{M}_{j,k}$ for entries whose signs are known. Notice that $\mathbf{Y}_{j,:}^{(-2)} \Sigma^{(1)} \widehat{w}^{(j)}$ has exactly the same formation of $\mathbf{Y}^{(-1)}$ and we can call Algorithm 1 to get its upper bound.

Finally, assume that from Algorithm 1 we already obtained $\{(\mathbf{L}^{(-l)}, \mathbf{U}^{(-l)})\}_{l=1}^H$, we summarize the algorithm to compute upper bound of L_∞^S in Algorithm 2.

Improve the bound for robustness verification. In some applications (e.g., robustness verification), we only need to bound $\|f(x) - f(s)\|$ for a fixed s and $x \in B[s;R]$. Although $L^{B[s;R]} \cdot R$ gives a bound of $\|f(x) - f(s)\|$, we can make this bound tighter by using an integral:

Theorem 2.

$$\|f(x) - f(s)\| \leqslant \int_0^R L^{B[s;t]} dt \leqslant L^{B[s;R]} \cdot R, \forall x \in B[s;R].$$

In practice, the integral $\int_0^R L^{B[s;t]} dt$ can be upper bounded by evaluating at n intervals:

$$\int_0^R L^{B[s;t]} dt \leqslant \sum_{i=1}^n L^{B[s;t_i]} \Delta t, \quad (24)$$

where we divide R into n segments $t_0 = 0, t_1, t_2, \cdots, t_{n-1}, t_n = R$, and $t_{i+1} - t_i = \Delta t$.

Applications and Experiments[1]

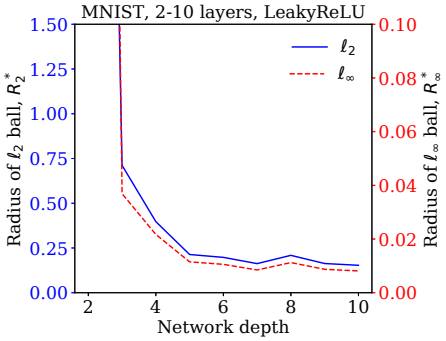
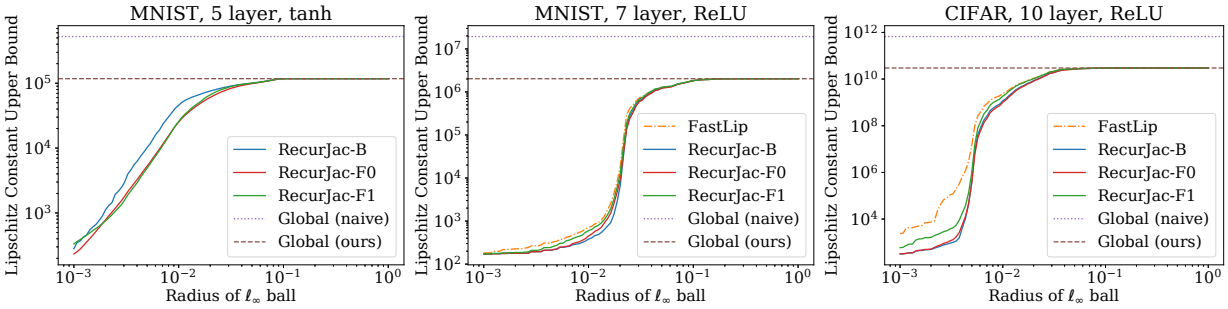


Figure 2: The largest radius R^* within which no stationary point exists, for network with different depths (2-10 layers)

Local optimization landscape. In non-convex optimization, a zero gradient vector results in a stationary point, potentially a saddle point or a local minimum. The existence of saddle points and local minima is one of the main difficulties for non-convex optimization (Dauphin et al. 2014),

[1]Source code for RecurJac and all experiments is available at http://github.com/huanzhang12/RecurJac-Jacobian-bounds



(a) MNIST 5-layer tanh activation (b) MNIST 7-layer ReLU activation (c) CIFAR 10-layer ReLU activation

Figure 3: Global and local Lipschitz constants on three networks. FastLip can only be applied to (leaky)ReLU networks.

		runner-up target		random target		least-likely target	
Network	Method	Undefended	Adv. Training	Undefended	Adv. Training	Undefended	Adv. Training
MNIST 3-layer	RecurJac	0.02256	**0.11573**	0.02870	**0.13753**	0.03205	**0.16153**
	FastLip	0.01802	0.09639	0.02374	0.11753	0.02720	0.14067
MNIST 4-layer	RecurJac	0.02104	**0.07350**	0.02399	**0.08603**	0.02519	**0.09863**
	FastLip	0.01602	0.04232	0.01882	0.05267	0.02018	0.06417

Table 1: Comparison of the lower bounds for ℓ_∞ distortion found by RecurJac (our algorithm) and FastLip on models with adversarial training with PGD perturbation $\epsilon = 0.3$ for two models and 3 targeted attack classes, averaged over 100 images.

including optimization problems on neural networks. However, if for at least one pair of $\{j, k\}$ we have $\mathbf{U}_{j,k} < 0$ or $\mathbf{L}_{j,k} > 0$, the Jacobian $\mathbf{Y}$ will never become a zero matrix within a local region.

In this experiment, we train an MLP network with leaky-ReLU activation ($\alpha = 0.3$) for MNIST and varying network depth from 2 to 10. Each hidden layer has 20 neurons, and all models achieve over 96% accuracy on validation set. We randomly choose 500 images of digit "1" from the test set that are correctly classified by all models, and bound the gradient of $f_1(x)$ (logit output for class "1"). For each image, we record the largest ℓ_2 and ℓ_∞ distortion (denoted as R_2^* and R_∞^*) added such that there is at least one element k in $\nabla f_1(x)$ that can never reach zero (i.e., $\mathbf{U}_{1,k} < 0$ or $\mathbf{L}_{1,k} > 0$). The reported R^* are the average of 500 images.

Figure 2 shows how R^* decreases as the network depth increases. Interestingly, for the smallest network with only 2 layers, no stationary point is found in its entire domain ($R^* = \infty$). For deeper networks, the region without stationary points near x becomes smaller, indicating the difficulty of finding optimal adversarial examples (a global optima with minimum distortion) grows with network depth.

Local and global Lipschitz constant. We apply RecurJac to get local and global Lipschitz constants on 4 networks of different scales for MNIST and CIFAR. For MNIST, we use a 10-layer leaky-ReLU network with 20 neurons per layer, a 5-layer tanh network with 50 neurons per layer, a 7-layer ReLU network with 1024, 512, 256, 128, 64 and 32 hidden neurons; for CIFAR, we use a 10-layer network with 2048, 2048, 1024, 1024, 512, 512, 256, 256, 128 hidden neurons.

As a comparison, we include Lipschitz constants computed by Fast-Lip (Weng et al. 2018a), a state-of-the-art algorithm for ReLU networks (we also trivially extended it to the leaky ReLU case for comparison). For our algorithm, we run both the backward and the forward versions, denoted as RecurJac-B (Algorithm 1) and RecurJac-F0 (the forward version). RecurJac-F0 requires to maintain intermediate bounds in shape $n_l \times n_0$, thus the computational cost is very high. We implemented another forward version, RecurJac-F1, which starts intermediate bounds after the first layer and reduce the space complexity to $n_l \times n_1$.

We randomly select an image for each dataset and as the input. Then, we upper bound the Local Lipschitz constant within an ℓ_∞ ball of radius R. As shown in Figure 1 and 3, for all networks, when R is small, our algorithms significantly outperforms Fast-Lip as we find much smaller (and thus in better quality) Lipschitz constants (sometimes a few magnitudes smaller, noting the logarithmic y-axis); When R is large, local Lipschitz constant converges to a value which corresponds to the worst case activation pattern, which is in fact a global Lipschitz constant. Although this value is large, it is still magnitudes smaller than the global Lipschitz constant obtained by the naive product of weight matrices' induced norms (dotted lines with label "naive").

For the largest CIFAR network, the average computation time for 1 local Lipschitz constant of FastLin, RecurJac-B, RecurJac-F0 and RecurJac-F1 are 2.4 sec, 10.5 sec, 1 hr and 5 hr respectively, on 1 CPU core. RecurJac-F0 and RecurJac-F1 sometimes provide better results than RecurJac-B (Fig. 3a). However when $n_H \ll n_0$, RecurJac-B is preferred due to its computational efficiency.

Robustness verification for adversarial examples. For a correctly classified source image s of class c and an attack target class j, we define $g(s) = f_c(s) - f_j(s) > 0$ that represents the margin between two classes. For $x \in B_{\ell_p}[s; R]$,

if $g(x)$ goes below 0, an adversarial example x is found. Using Theorem 2, we know that the largest R such that $\int_0^R L^{B[s;t]}(g)dt < g(s)$ is a certified robustness lower bound within which no adversarial examples of class j can be found. In this experiment, we approximate the integral in (24) from above by using 30 intervals.

We evaluate the robustness lower bound on undefended networks and adversarially trained networks proposed by (Madry et al. 2018). We use two MLP networks with 3 and 4 layers with 1024 neurons per layer. Table 1 shows that our algorithm can indeed reflect the increased robustness as the certified lower bounds under "Adv. Training" column become much larger than "Undefended". Additionally, when the adversarial training procedure attempts to defend against adversarial examples with ℓ_∞ distortion less than 0.3, our bounds are better than Fast-Lip and closer to 0.3, suggesting that adversarial training is effective.

Conclusion

In this paper, we propose a novel algorithm, RecurJac, for recursively bounding a neural network's Jacobian matrix with respect to its input. Our method can be efficiently applied to networks with a wide class of activation functions. Applications of RecurJac include characterizing local optimization landscape, computing a local or global Lipschitz constant, and robustness verification of neural networks.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein generative adversarial networks. In *ICML*, 214–223.

Arora, S., and Zhang, Y. 2018. Do GANs actually learn the distribution? an empirical study. *ICLR*.

Bartlett, P. L.; Foster, D. J.; and Telgarsky, M. J. 2017. Spectrally-normalized margin bounds for neural networks. In *NIPS*, 6240–6249.

Cisse, M.; Bojanowski, P.; Grave, E.; Dauphin, Y.; and Usunier, N. 2017. Parseval networks: Improving robustness to adversarial examples. *ICML*.

Croce, F.; Andriushchenko, M.; and Hein, M. 2018. Provable robustness of ReLU networks via maximization of linear regions. *arXiv preprint arXiv:1810.07481*.

Dauphin, Y. N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; and Bengio, Y. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2933–2941.

De Haan, L., and Ferreira, A. 2007. *Extreme value theory: an introduction*. Springer Science & Business Media.

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

Elsayed, G. F.; Krishnan, D.; Mobahi, H.; Regan, K.; and Bengio, S. 2018. Large margin deep networks for classification. *arXiv preprint arXiv:1803.05598*.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *NIPS*, 2672–2680.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of Wasserstein GANs. *NIPS*.

Hein, M., and Andriushchenko, M. 2017. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2266–2276.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.

Mirman, M.; Gehr, T.; and Vechev, M. 2018. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, 3575–3583.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Qian, H., and Wegman, M. N. 2018. L2-nonexpansive neural networks. *arXiv preprint arXiv:1802.07896*.

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

Sriperumbudur, B. K.; Fukumizu, K.; Gretton, A.; Schölkopf, B.; and Lanckriet, G. R. 2009. On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*.

Tsuzuku, Y.; Sato, I.; and Sugiyama, M. 2018. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *arXiv preprint arXiv:1802.04034*.

Vapnik, V. N., and Vapnik, V. 1998. *Statistical learning theory*, volume 1. Wiley New York.

Weng, T.-W.; Zhang, H.; Chen, H.; Song, Z.; Hsieh, C.-J.; Boning, D.; Dhillon, I. S.; and Daniel, L. 2018a. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning*.

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

Wong, E., and Kolter, Z. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 5283–5292.

Wong, E.; Schmidt, F.; Metzen, J. H.; and Kolter, J. Z. 2018. Scaling provable adversarial defenses. *NIPS*.

Wood, G., and Zhang, B. 1996. Estimation of the Lipschitz constant of a function. *Journal of Global Optimization* 8(1):91–103.

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

Zhang, P.; Liu, Q.; Zhou, D.; Xu, T.; and He, X. 2018b. On the discrimination-generalization tradeoff in GANs. *ICLR*.