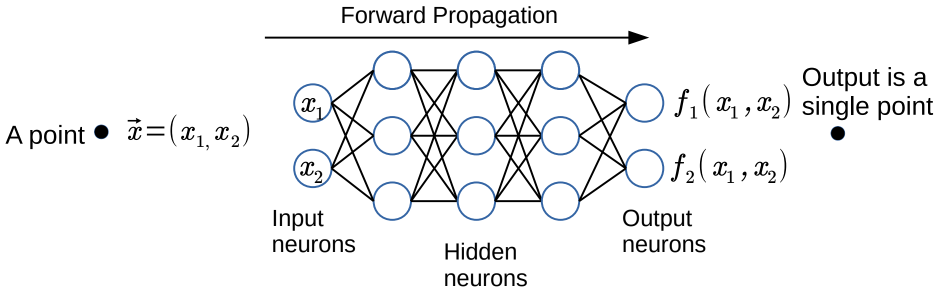
RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks

Huan Zhang (**UCLA**)
Pengchuan Zhang (**Microsoft Research**)
Cho-Jui Hsieh (**UCLA**)

Slides, paper and code: `github.com/huanzhang12/RecurJac`

Verification of Neural Networks

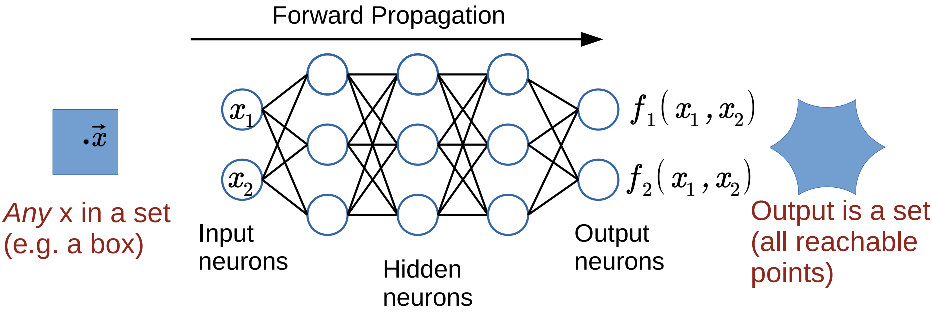
Inference in neural networks



$$y = f(x)$$

Verification of Neural Networks

When the input is not a single point...

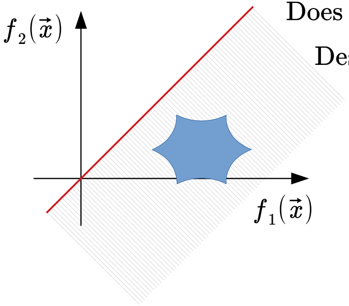


$$x \in S$$

$$y \in \{f(x), x \in S\}$$

Verification of Neural Networks

The verification problem (e.g., margin between two classes)



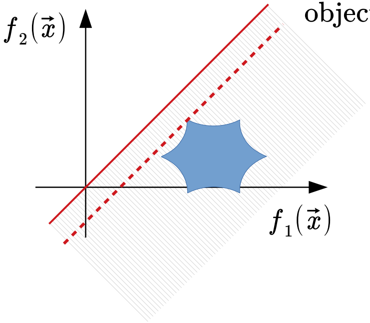
Does $f_1(\vec{x}) > f_2(\vec{x})$ always hold?

Design an objective g for it

$$\begin{aligned} \min \quad & g(y) \\ \text{s.t.} \quad & x \in S \\ & y \in \{f(x), x \in S\} \end{aligned}$$

Verification of Neural Networks

Solving the verification problem exactly

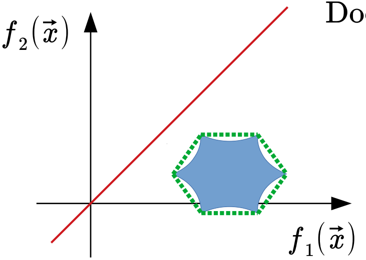


objective $g \overset{\text{def}}{=} f_1(\vec{x}) - f_2(\vec{x}) > 0$

$$
\begin{aligned}
\min \quad & g(y) \\
\text{s.t.} \quad & x \in S \\
& y \in \{f(x), x \in S\}
\end{aligned}
$$

Verification of Neural Networks - Convex Relaxations

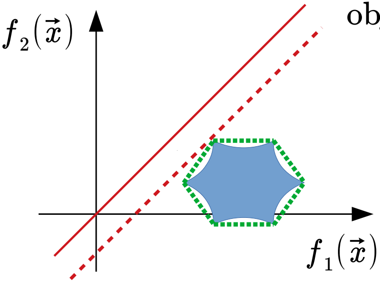
Solving the verification problem through convex relaxations



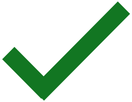
Does $f_1(\vec{x}) > f_2(\vec{x})$ always hold?

$$\begin{aligned} \min \quad & g(y) \\ \text{s.t.} \quad & x \in S \\ & y \in \text{Convex}(\{f(x), x \in S\}) \end{aligned}$$

Solving the verification problem through convex relaxations (ideal case)

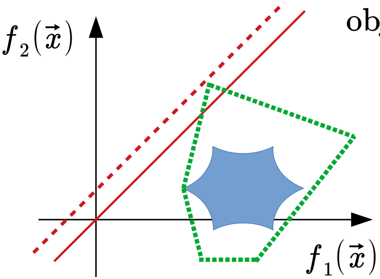


objective $g \stackrel{\text{def}}{=} f_1(\vec{x}) - f_2(\vec{x}) > 0$



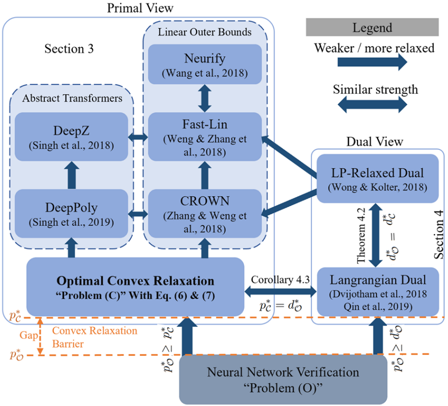
$$\begin{aligned} \min \quad & g(y) \\ \text{s.t.} \quad & x \in S \\ & y \in \text{Convex}(\{f(x), x \in S\}) \end{aligned}$$

Solving the verification problem through convex relaxations (the reality)



objective $g \stackrel{\text{def}}{=} f_1(\vec{x}) - f_2(\vec{x}) < 0$

Sound, but not complete



Original problem:

$$\begin{aligned} &\min \quad g(y) \\ &\text{s.t.} \;\; x \in S \\ &\qquad y \in \{f(S)\} \end{aligned}$$

Convex relaxed:

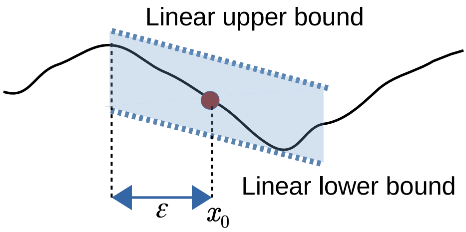
$$\begin{aligned} &\min \quad g(y) \\ &\text{s.t.} \;\; x \in S \\ &y \in \text{Convex}(\{f(S)\}) \end{aligned}$$

- Many verification methods can be seen in a unified framework as convex relaxations of the original non-convex verification problem

"A convex relaxation barrier to tight robustness verification of neural networks", Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh and Pengchuan Zhang, arXiv 1902.08722

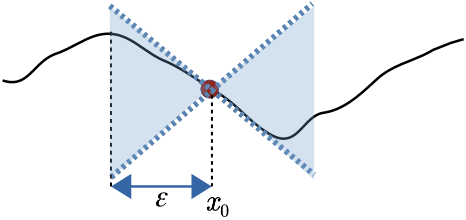
Verification of Neural Networks - "Greedy" Solvers

- Even solving the convex-relaxed problem can still be expensive; many "greedy" solvers are proposed (Neurify w/o BaB, CROWN, Fast-Lin, DeepZ, DeepPoly)
- These greedy solvers give linear outer bounds of NN w.r.t. all inputs $x_0 + \Delta x \in S$: $A_L \Delta x + b_L \leq f(x_0 + \Delta x) \leq A_U \Delta x + b_U$
- For example, one dimensional case, ℓ_∞ ball at x_0 with radius ε:

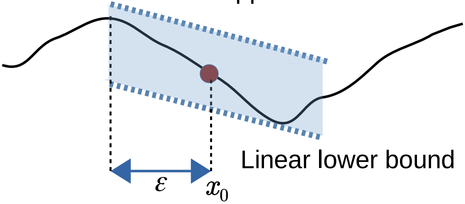


"A convex relaxation barrier to tight robustness verification of neural networks", Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh and Pengchuan Zhang, arXiv 1902.08722

In RecurJac, we use a different kind of bound: the Local Lipschitz constant based verification bound



Slope ±L: Local Lipschitz constant



Local Lipschitz Constant based Bounds

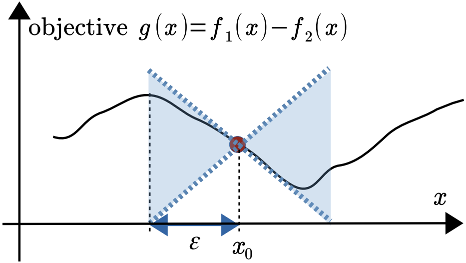
- A local Lipschitz constant is a scalar $L_{p,S}$ that satisfies

  $|g(x_1) - g(x_2)| \leq L_{p,S}\|x_1 - x_2\|_p$, for all $x_1, x_2 \in S := \{x | \|x - x_0\|_p \leq \varepsilon\}$

- We can lower bound a function $g(x)$ for all $x \in S$ by its local Lipschitz constant $L_{p,S}$:

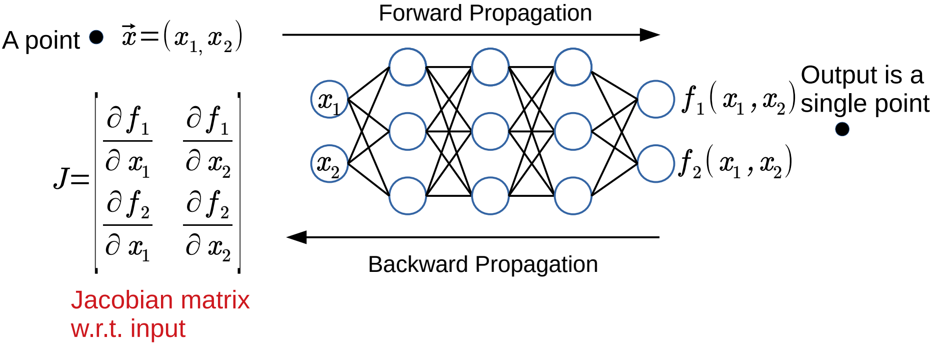
$$g(x) \geq g(x_0) - L_{p,S}\|x - x_0\|_p$$

- For example, $g(x) = f_1(x) - f_2(x)$ (margin) as we showed before

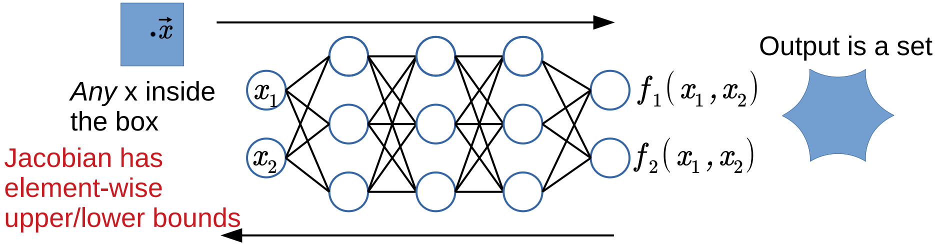


Verification of Neural Networks

Lipschitz constant is closely related to gradients. We thus look into the back-propagation in neural networks



Verification problem for Jacobian/Gradients



$$\begin{bmatrix} L_{1,1} & L_{1,2} \\ L_{2,1} & L_{2,2} \end{bmatrix} \leq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \leq \begin{bmatrix} U_{1,1} & U_{1,2} \\ U_{2,1} & U_{2,2} \end{bmatrix}$$

$$L \leqslant J \leqslant U$$

The RecurJac Algorithm

- RecurJac gives element-wise bounds **U** and **L** for Jacobian $J := \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ of input x, not network weights $\boldsymbol{w}$ (not a bound for $\nabla_{\boldsymbol{w}} f(\boldsymbol{x}, \boldsymbol{w})$)
- RecurJac can be applied to networks with any common activation functions (tanh, sigmoid, etc), not limited to ReLU
- RecurJac is polynomial time – its time complexity is $O(H^2 n^3)$ for a H-layer network with n neurons per layer

From Jacobian Bounds to Local Lipschitz Constant

- Local Lipschitz constant can be seen as the maximum induced $p-$norm of Jacobian matrix for all $x \in S$:

$$L_{p,S} = \max_{x \in S} \|J\|_p$$

- With $\mathbf{U}$ and $\mathbf{L}$ where $\mathbf{L} \leq \nabla f(x) \leq \mathbf{U}$ for $x \in S$, an (upper bound of) Local Lipschitz constant can be easily obtained.
- Define matrix M with each element $M_{i,j} = \max(|\mathbf{L}_{i,j}|, |\mathbf{U}_{i,j}|)$, then

$$L_{p,S} \leq \|M\|_p$$

The RecurJac Algorithm

- For illustration, for a 3-layer neural network with ReLU activation σ:

$$f(\boldsymbol{x}) = \mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\boldsymbol{x}))$$

where

$$\mathbf{W}^{(3)} = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$$

- the element $\{j, k\}$ in Jacobian matrix can be written as:

$$J_{j,k} = [\nabla f_j(\boldsymbol{x})]_k = \mathbf{W}^{(3)}_{j,:}\Sigma^{(2)}\mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}_{:,k}$$

where

$$\Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \Sigma^{(1)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix}$$

- "?" can be 0 or 1, reflecting the state of a ReLU neuron
- We need to consider the worst case ReLU states to get bounds

The RecurJac Algorithm

- For illustration, for a 3-layer neural network with ReLU activation σ:

$$f(\boldsymbol{x}) = \mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\boldsymbol{x}))$$

where

$$\mathbf{W}^{(3)} = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$$

- the element $\{j, k\}$ in Jacobian matrix can be written as:

$$J_{j,k} = [\nabla f_j(\boldsymbol{x})]_k = \mathbf{W}^{(3)}_{j,:}\Sigma^{(2)}\mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}_{:,k}$$

where

$$\Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \Sigma^{(1)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix}$$

- "?" can be 0 or 1, reflecting the state of a ReLU neuron
- We need to consider the worst case ReLU states to get bounds

Basic Ideas Behind RecurJac

- Define $\mathbf{Y}^{(1)} = \mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, where

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \quad \Sigma^{(1)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$$

- Rule: choose worst case "?" to maximize and minimize each element
- $\mathbf{U}^{(1)}_{1,1} = 1 \times 1 \times 1 + 1 \times 1 \times 2 = 3$
- $\mathbf{L}^{(1)}_{1,1} = 1 \times 0 \times 1 + 1 \times 0 \times 2 = 0$

Basic Ideas Behind RecurJac

- Define $\mathbf{Y}^{(1)} = \mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, where

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \quad \Sigma^{(1)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$$

- Rule: choose worst case "?" to maximize and minimize each element
- $\mathbf{U}^{(1)}_{1,1} = 1 \times 1 \times 1 + 1 \times 1 \times 2 = 3$
- $\mathbf{L}^{(1)}_{1,1} = 1 \times 0 \times 1 + 1 \times 0 \times 2 = 0$
- $\mathbf{U}^{(1)}_{1,2} = 1 \times 0 \times (-1) + 1 \times 1 \times 1 = 1$
- $\mathbf{L}^{(1)}_{1,2} = 1 \times 1 \times (-1) + 1 \times 0 \times 1 = -1$

- Define $\mathbf{Y}^{(1)} = \mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, where
$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \quad \Sigma^{(1)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$$
- Rule: choose worst case "?" to maximize and minimize each element
- $\mathbf{U}^{(1)}_{1,1} = 1 \times 1 \times 1 + 1 \times 1 \times 2 = 3$
- $\mathbf{L}^{(1)}_{1,1} = 1 \times 0 \times 1 + 1 \times 0 \times 2 = 0$
- $\mathbf{U}^{(1)}_{1,2} = 1 \times 0 \times (-1) + 1 \times 1 \times 1 = 1$
- $\mathbf{L}^{(1)}_{1,2} = 1 \times 1 \times (-1) + 1 \times 0 \times 1 = -1$
- $\mathbf{U}^{(1)} = \begin{bmatrix} 3 & 1 \\ 4 & 1 \end{bmatrix} \quad \mathbf{L}^{(1)} = \begin{bmatrix} 0 & -1 \\ 0 & -2 \end{bmatrix}$

Basic Ideas Behind RecurJac - The Next Layer

- Define $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, $\mathbf{U}^{(2)} \leq \mathbf{Y}^{(2)} \leq \mathbf{L}^{(2)}$, where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{L}^{(1)} = \begin{bmatrix} \mathbf{0} & -1 \\ \mathbf{0} & -2 \end{bmatrix} \quad \mathbf{U}^{(1)} = \begin{bmatrix} \mathbf{3} & 1 \\ \mathbf{4} & 1 \end{bmatrix}$$

- Rule: choose worst case "?" and the worst case numbers from $\mathbf{U}^{(1)}$ and $\mathbf{L}^{(2)}$ to maximize and minimize each element.
- For example:

$$\begin{aligned} \mathbf{U}^{(2)}_{1,1} &= 1 \times \{1, 0\} \times [0, 3] + (-1) \times \{1, 0\} \times [0, 4] \\ &= 1 \times 1 \times 3 + (-1) \times 0 \times 0 \\ &= 3 \end{aligned}$$

Basic Ideas Behind RecurJac - The Next Layer

- Define $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, $\mathbf{U}^{(2)} \leq \mathbf{Y}^{(2)} \leq \mathbf{L}^{(2)}$, where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{L}^{(1)} = \begin{bmatrix} \mathbf{0} & -1 \\ \mathbf{0} & -2 \end{bmatrix} \quad \mathbf{U}^{(1)} = \begin{bmatrix} \mathbf{3} & 1 \\ \mathbf{4} & 1 \end{bmatrix}$$

- Rule: choose worst case "?" and the worst case numbers from $\mathbf{U}^{(1)}$ and $\mathbf{L}^{(2)}$ to maximize and minimize each element.
- For example:

$$\begin{aligned} \mathbf{U}^{(2)}_{1,1} &= 1 \times \{1, 0\} \times [0, 3] + (-1) \times \{1, 0\} \times [0, 4] \\ &= 1 \times 1 \times 3 + (-1) \times 0 \times 0 \\ &= 3 \end{aligned}$$

- This is the "Fast-Lip" algorithm (Weng and Zhang et al., ICML 2018)

Basic Ideas Behind RecurJac - The Next Layer

- Define $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, $\mathbf{U}^{(2)} \leq \mathbf{Y}^{(2)} \leq \mathbf{L}^{(2)}$, where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{L}^{(1)} = \begin{bmatrix} \mathbf{0} & -1 \\ \mathbf{0} & -2 \end{bmatrix} \quad \mathbf{U}^{(1)} = \begin{bmatrix} \mathbf{3} & 1 \\ \mathbf{4} & 1 \end{bmatrix}$$

- Rule: choose worst case “?” <u>and</u> the worst case numbers from $\mathbf{U}^{(1)}$ and $\mathbf{L}^{(2)}$ to maximize and minimize each element.
- Important observation: $\mathbf{L}^{(1)}_{1,1} = 0$, $\mathbf{L}^{(1)}_{2,1} = 0$, so $\mathbf{Y}^{(1)}_{1,1} \geq 0$, $\mathbf{Y}^{(1)}_{2,1} \geq 0$
- In this case, no matter what the values $\mathbf{Y}^{(1)}_{1,1}$, $\mathbf{Y}^{(1)}_{2,1}$ take, “?” are always fixed to $\Sigma^{(2)} = \begin{bmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{0} \end{bmatrix}$ compute $\mathbf{U}^{(2)}_{1,1}$

Basic Ideas Behind RecurJac - The Next Layer

- Define $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)}$, element-wise bound: $\mathbf{U}^{(1)} \leq \mathbf{Y}^{(1)} \leq \mathbf{L}^{(1)}$, $\mathbf{U}^{(2)} \leq \mathbf{Y}^{(2)} \leq \mathbf{L}^{(2)}$, where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} ? & 0 \\ 0 & ? \end{bmatrix} \quad \mathbf{L}^{(1)} = \begin{bmatrix} \mathbf{0} & -1 \\ \mathbf{0} & -2 \end{bmatrix} \quad \mathbf{U}^{(1)} = \begin{bmatrix} \mathbf{3} & 1 \\ \mathbf{4} & 1 \end{bmatrix}$$

- Rule: choose worst case "?" and the worst case numbers from $\mathbf{U}^{(1)}$ and $\mathbf{L}^{(2)}$ to maximize and minimize each element.
- Important observation: $\mathbf{L}^{(1)}_{1,1} = 0$, $\mathbf{L}^{(1)}_{2,1} = 0$, so $\mathbf{Y}^{(1)}_{1,1} \geq 0$, $\mathbf{Y}^{(1)}_{2,1} \geq 0$
- In this case, no matter what the values $\mathbf{Y}^{(1)}_{1,1}$, $\mathbf{Y}^{(1)}_{2,1}$ take, "?" are always fixed to $\Sigma^{(2)} = \begin{bmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{0} \end{bmatrix}$ compute $\mathbf{U}^{(2)}_{1,1}$
- $\Sigma^{(2)} = \begin{bmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{0} \end{bmatrix}$ with uncertain "?"s removed!

- Now propagate to the previous layer to refine the bound
- $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}$
- define $\tilde{\mathbf{W}}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{W}^{(2)}$ where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{0} \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

- Now run RecurJac on the equivalent 2-layer network $\tilde{\mathbf{W}}^{(2)}_{1,:}\Sigma^{(1)}\mathbf{W}^{(1)}_{:,1}$ and obtain its upper bound, which becomes $\mathbf{U}^{(2)}_{1,1}$
- this upper bound (calculated as 1) is tighter than 3 (by Fast-Lip)

Basic Ideas Behind RecurJac - Bound Propagation

- Now propagate to the previous layer to refine the bound
- $\mathbf{Y}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{Y}^{(1)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{W}^{(2)}\Sigma^{(1)}\mathbf{W}^{(1)}$
- define $\tilde{\mathbf{W}}^{(2)} = \mathbf{W}^{(3)}\Sigma^{(2)}\mathbf{W}^{(2)}$ where

$$\mathbf{W}^{(3)} = \begin{bmatrix} \mathbf{1} & -\mathbf{1} \end{bmatrix} \quad \Sigma^{(2)} = \begin{bmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{0} \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

- Now run RecurJac on the equivalent 2-layer network $\tilde{\mathbf{W}}^{(2)}_{1,:}\Sigma^{(1)}\mathbf{W}^{(1)}_{:,1}$ and obtain its upper bound, which becomes $\mathbf{U}^{(2)}_{1,1}$
- For network with l layers, RecurJac will form equivalent networks with $l-1$ layers and call the RecurJac procedure recursively for bound refinement, until reaching the base case of a 2-layer network

Basic Ideas Behind RecurJac

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)

  $(\mathbf{L}^{(-l-1,-l)}, \backsim) = \text{ComputeLU}(\ \widecheck{\mathbf{W}}^{(l+1,l)}, \{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=l+2}^{H}, \ \{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=l+2}^{H})$

  ▷ Recursive call

  $(\backsim, \mathbf{U}^{(-l-1,-l)}) = \text{ComputeLU}(\ \widehat{\mathbf{W}}^{(l+1,l)}, \{(\mathbf{L}^{(-i)}, \mathbf{U}^{(-i)}, \mathbf{W}^{(i)})\}_{i=l+2}^{H}, \ \{\boldsymbol{l}'^{(i-1)}, \boldsymbol{u}'^{(i-1)}\}_{i=l+2}^{H})$

  ▷ Recursive call

  Compute $\mathbf{L}^{(-l),\pm}$ from (14), $\mathbf{U}^{(-l),\pm}$ from (15)

  $\mathbf{L}^{(-l)} = \mathbf{L}^{(-l),\pm} + \mathbf{L}^{(-l-1,-l)}$

  $\mathbf{U}^{(-l)} = \mathbf{U}^{(-l),\pm} + \mathbf{U}^{(-l-1,-l)}$

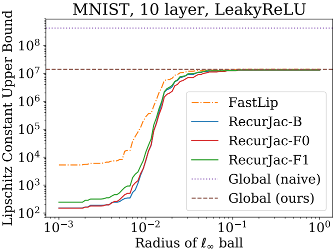
end if

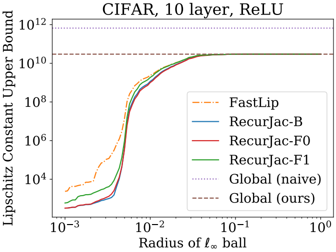
return $\mathbf{L}^{(-l)}$, $\mathbf{U}^{(-l)}$

For computing the Jacobian bounds:

- Fast-Lip similar to IBP (interval bound propagation)
- RecurJac similar to symbolic propagation (Fast-Lin, Neurify, CROWN)

How does RecurJac perform?





- Bounds on local Lipschitz constants are a few magnitudes better than Fast-Lip
- Bounds on global Lipschitz constants are a few magnitudes better than the product of operator norms for each layer

How does RecurJac perform?

RecurJac can find larger (thus better) robustness lower bound than previous Lipschitz constant bound based method, Fast-Lip.

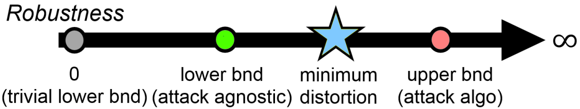


Table: Comparison of the lower bounds for ℓ_∞ distortion found by RecurJac and FastLip on models with adversarial training with PGD perturbation $\epsilon = 0.3$ for two models and 3 targeted attack classes, averaged over 100 images.

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

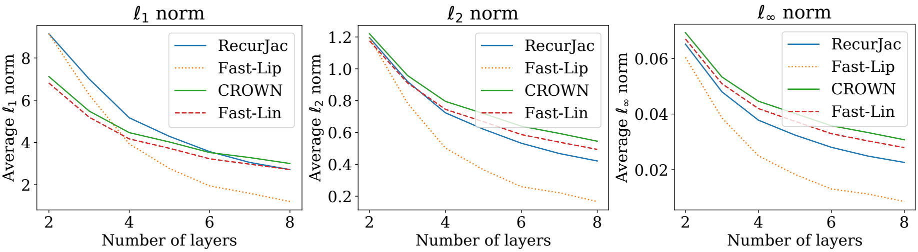
RecurJac vs. CROWN vs. Fast-Lin vs. Fast-Lip

Baselines:

- **RecurJac** (Zhang et al., AAAI 2019): improved Jacobian bound with recursive bound refinements
- **CROWN** (Zhang and Weng et al., NIPS 2018): Improved Fast-Lin, general upper and lower bounds, general activation functions (sigmoid, etc)
- **Fast-Lin** (Weng and Zhang et al., ICML 2018): linear upper and lower bounds with the same slope for ReLU, similar to Neurify (w/o BaB)
- **Fast-Lip** (Weng and Zhang et al., ICML 2018): first Jacobian bound

RecurJac vs. CROWN vs. Fast-Lin vs. Fast-Lip

- MNIST MLP with 2-8 layers, ReLU, 100 neurons per layer
- reported numbers are the certified lower bounds of minimum adversarial distortions (larger is better), averaged over 100 images
- input distortion is bounded by ℓ_1, ℓ_2 or ℓ_∞ norms



Other Verification Situations

- RecurJac can be used for verifying NN properties involving gradients or Jacobian, for example, the "first order" verification problem below:

$$\begin{aligned} \min \quad & c^\top \nabla f(x) \\ \text{s.t.} \quad & x \in S \\ & y = f(x) \end{aligned}$$

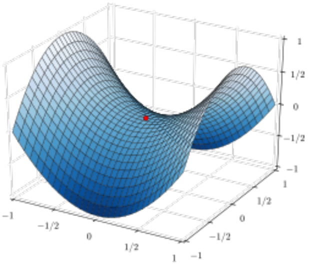
- But so far I haven't demonstrated a practical use of this (yet)

Beyond Verification

- Bounds on Jacobian can be useful for applications beyond verification
- Local Lipschitz constant is also a surrogate to many other problems (e.g., Wasserstein GANs, some generalization bounds, etc)
- Gain understanding on optimization landscape

Beyond Verification - Optimization Landscape

- A stationary point in optimization is a $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) = \mathbf{0}$.
- Gradient based optimization converges to stationary points (global minima, local minima or saddle points).



- RecurJac bound gives us:

$$\mathbf{L}_{j,k} \leq [\nabla f_j(\boldsymbol{x})]_k \leq \mathbf{U}_{j,k} \quad \forall j,k.$$

- If $\mathbf{L}_{j,k} > 0$ or $\mathbf{U}_{j,k} < 0$, we know that $[\nabla f_j(\boldsymbol{x})]_k$ never becomes 0 for all $\boldsymbol{x} \in S$, so no stationary points exist inside S

Beyond Verification - Optimization Landscape

Using RecurJac, we can get a rough idea on optimization landscape w.r.t. input by looking at the radius where no stationary points exist:

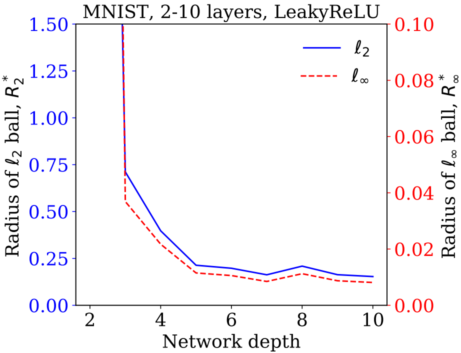


Figure: The largest radius R^* (centered at a test example) within which no stationary point exists, for network with different depths (2-10 layers), averaged over 100 test examples

Interestingly, for network of 2 layers, the radius is infinity.

Thank You!

Questions?

Slides, paper and source code available at
github.com/huanzhang12/RecurJac