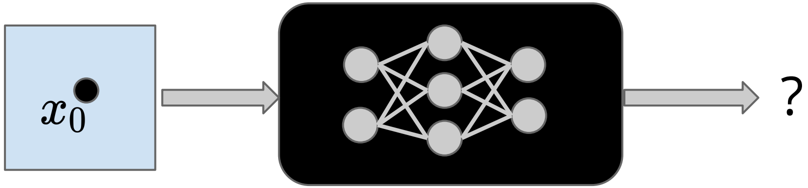
AAAI 2022 Tutorial on Neural Network Verification

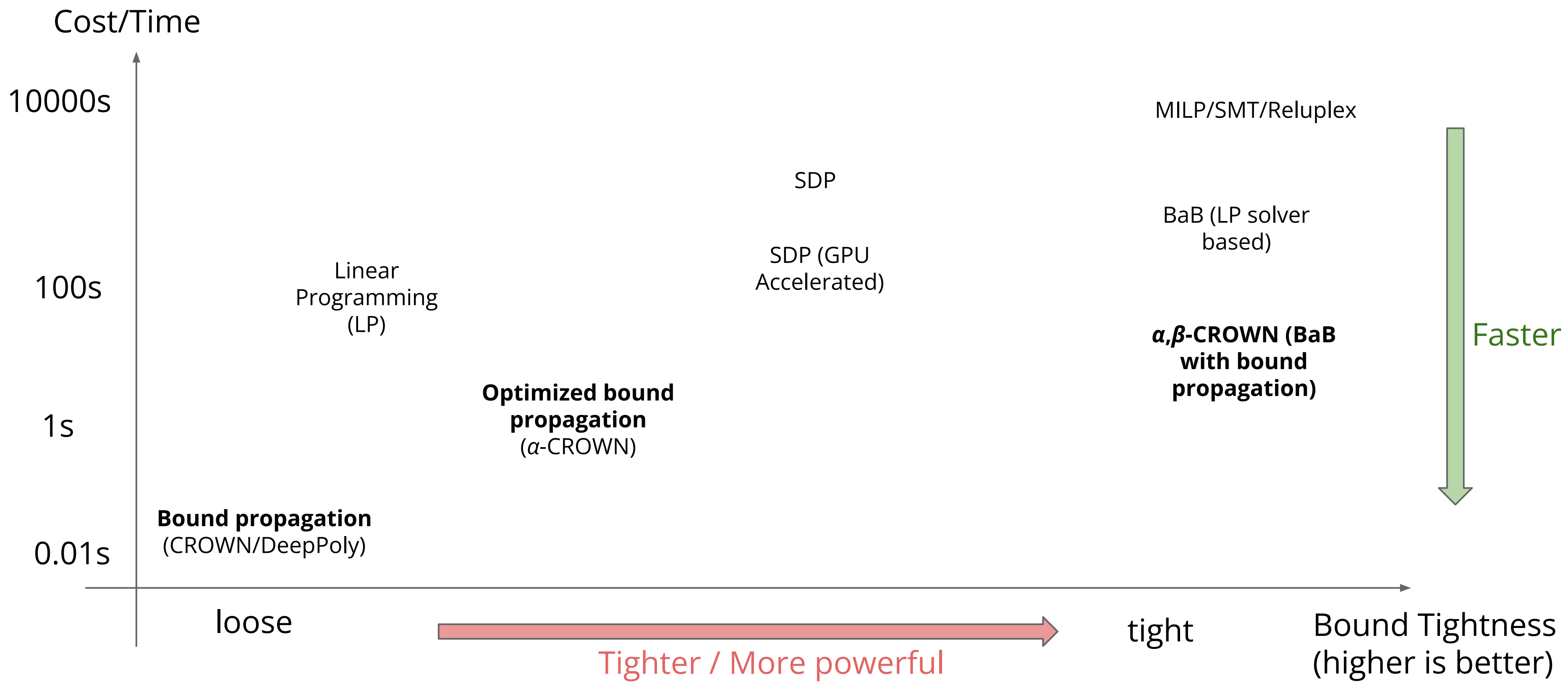
Part II: Algorithms for NN Verification

Huan Zhang (CMU), Kaidi Xu (Drexel), Shiqi Wang (Columbia) and Cho-Jui Hsieh (UCLA)

Feb 23, 2022



Neural Network Verification: Representative Algorithms



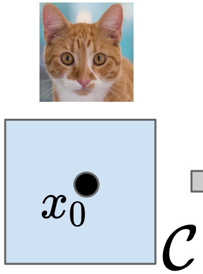
Today's Focus

- **Bound propagation** based verifiers (CROWN/DeepPoly)
  - not relying on a LP/MIP solver
  - GPU accelerated, scalable to large networks
- **Bound optimization** to achieve a tighter bound
  - α-CROWN: Cheaply optimizable bounds using gradient ascent
- **Branch and bound** (from incomplete to complete)
  - β-CROWN: bound propagation with branch and bound (**winner of VNN-COMP 2021, highest score among 11 tools**)

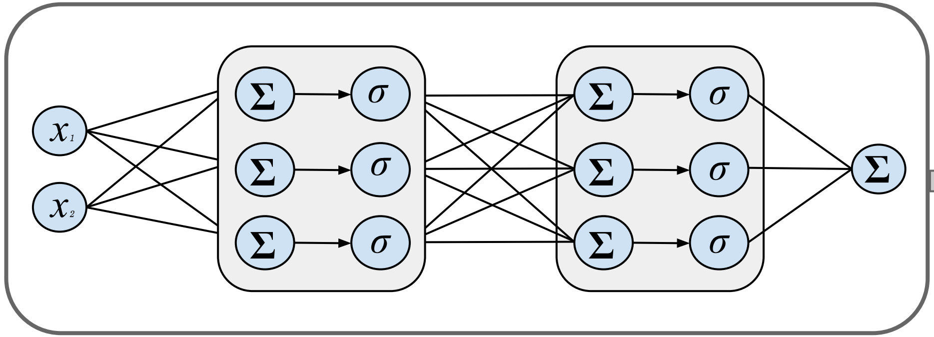
Quick Recap: The Basic Formulation

Suppose $f(x_0) > 0$. Can we verify this property:

$$f(x) > 0, \forall x \in \mathcal{C}$$



Input is a set



Neural Network

Output is a range/set of scores

$$\boxed{0.2} \leq f(x) \leq 2.2$$

"cat" even in the worst case

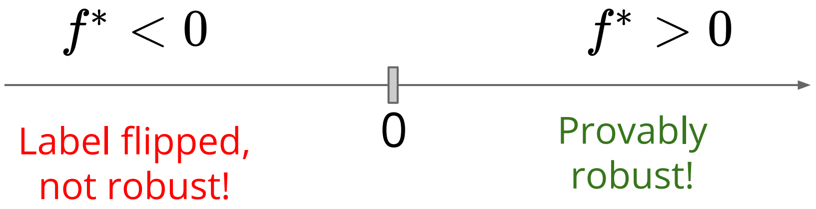
Must consider a set of infinite points as the input of the NN.

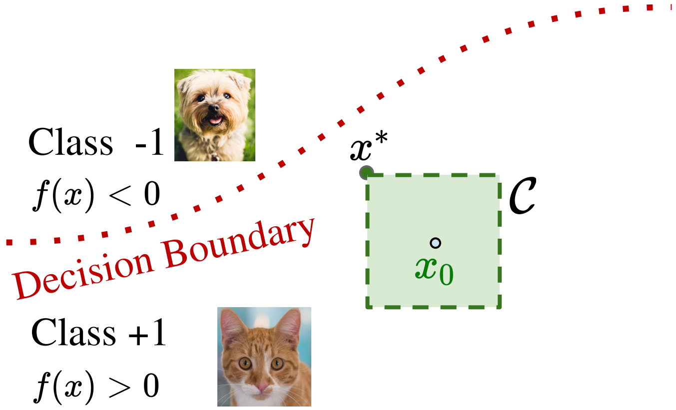
Quick Recap: The Basic Formulation

Assuming $f(x_0) > 0$, we solve the optimization problem to find the worst case:

$$f^* = \min_{x \in \mathcal{C}} f(x)$$

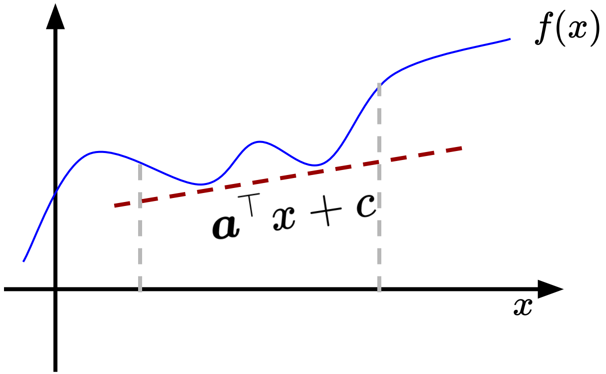
$\mathcal{C}$ is usually a perturbation set "around" x_0, e.g., $\mathcal{C} := \{x \mid \|x - x_0\|_p \leq \epsilon\}$





Outline: Algorithms for Neural Network Verification

CROWN:
bound propagation
based verification
(Zhang et al. NIPS 2018)



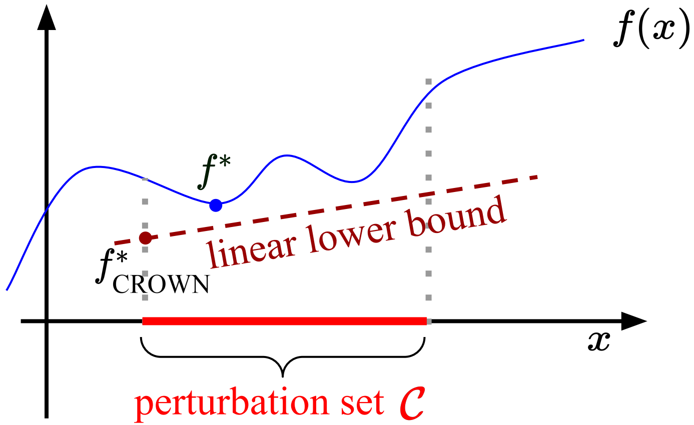
[1] Efficient Neural Network Robustness Certification with General Activation Functions, **Huan Zhang***, Tsui-Wei Weng*, Pin-Yu Chen, Cho-Jui Hsieh, Luca Daniel. **NIPS 2018**.
[2] Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. Kaidi Xu*, Zhouxing Shi*, **Huan Zhang***, Yihan Wang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, Cho-Jui Hsieh. **NeurIPS 2020**.
[3] Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers, Kaidi Xu*, **Huan Zhang***, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, Cho-Jui Hsieh. ICLR 2021.
[4] Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. Shiqi Wang*, Huan Zhang*, Kaidi Xu*, Xue Lin, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter. NeurIPS 2021.

(*Co-first authors).

CROWN: Bound Propagation based ImplVerification

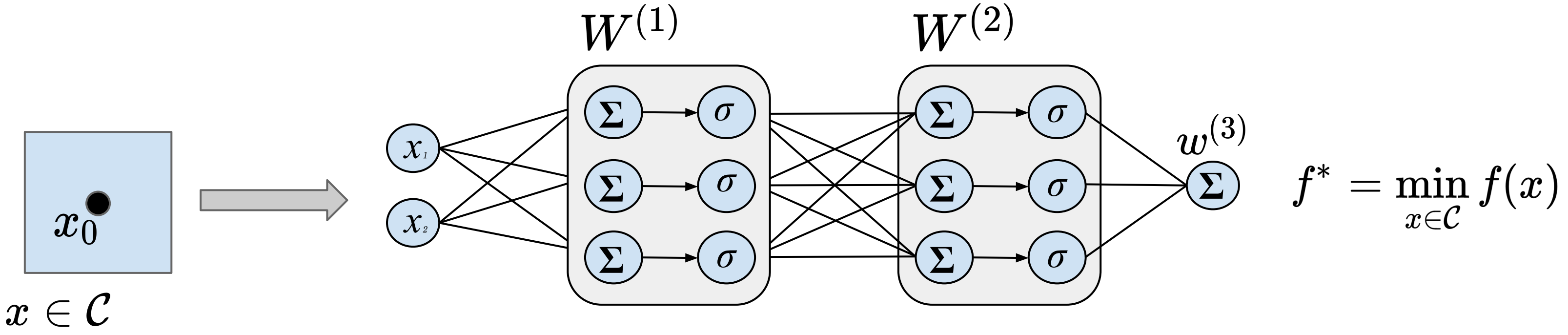
- We want to find a lower bound for this problem efficiently:

$$f^*_{\text{CROWN}} \leq f^* = \min_{x \in \mathcal{C}} f(x)$$



- $f^*_{\text{CROWN}} > 0 \Rightarrow f^* > 0$, so no adversarial example exists if $f^*_{\text{CROWN}} > 0$
- **CROWN** (Zhang et al. 2018) is an efficient linear **bound propagation** based algorithm to find linear lower/upper bounds of NNs
- Equivalent to **DeepPoly** (Singh et al., 2019), another popular verification algorithm

Find the Lower Bound on Feed-forward Networks



- If there are no non-linear operations (e.g., ReLUs), all weights can be multiplied together

$$f(x) = w^{(3)\top} W^{(2)} W^{(1)} x = a^\top x$$

- Bounds for linear functions are easy (e.g., Hölder's inequality for Lp norm)

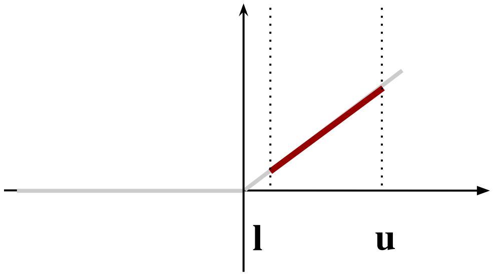
$$f^* := -\epsilon \|a\|_1 + a^\top x_0 \qquad x \in \{x \,|\, \|x - x_0\|_\infty \leq \epsilon\}$$

How to "Convert" ReLU into a Linear Function?

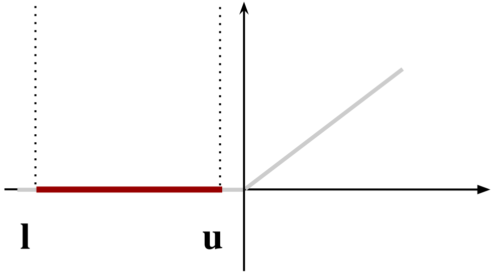
$$f(x) = w^{(3)}\mathrm{ReLU}(W^{(2)}\mathrm{ReLU}(W^{(1)}x))$$

$$\mathrm{ReLU}(z) = \max(0, z)$$

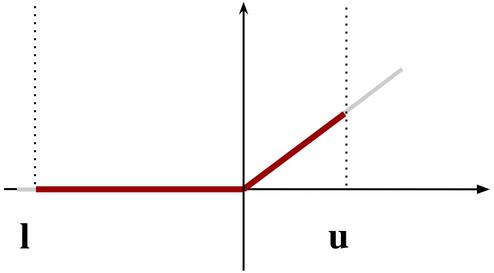
ReLU neurons have three cases depending on bounds on their inputs:



$\mathbf{l} \geq 0$, always active
(linear)



$\mathbf{u} \leq 0$, Always inactive
(zero)



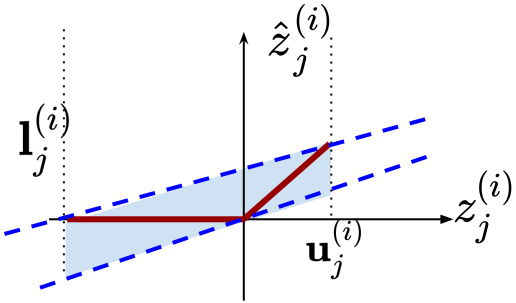
$\mathbf{l} \leq 0 \leq \mathbf{u}$
Unstable (non-linear)
Must be relaxed

$\mathbf{l}$ and $\mathbf{u}$ are pre-activation bounds (also called intermediate layer bounds)

How to "Convert" ReLU into a Linear Function?

For the j-th ReLU neuron in layer i: $\hat{z}_j^{(i)} = \mathrm{ReLU}\left(z_j^{(i)}\right)$

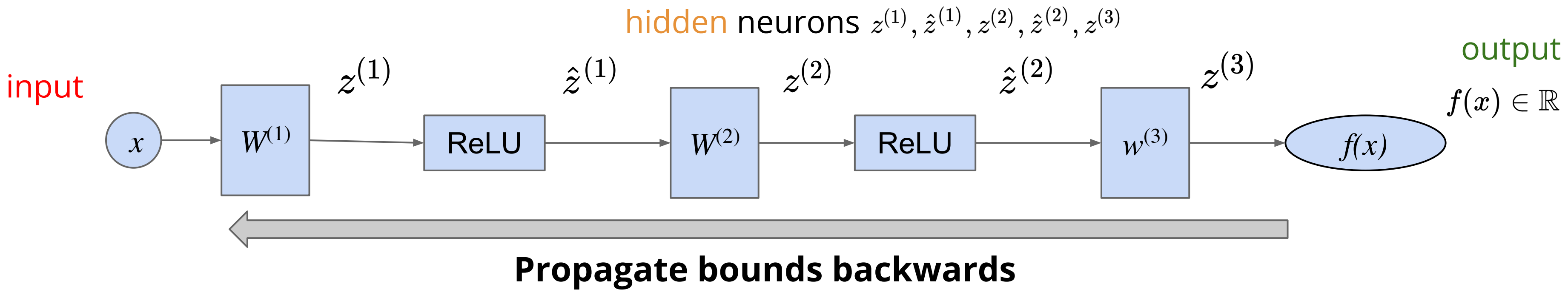
Assuming its input is bounded: $\mathbf{l}_j^{(i)} \leq z^{(i)} \leq \mathbf{u}_j^{(i)}$ and unstable: $\mathbf{l}_j^{(i)} \leq 0 \leq \mathbf{u}_j^{(i)}$



- Idea: use two linear bounds to replace ReLU, to obtain linear bounds for the entire network
- Can also be extended to non-ReLU functions (e.g., tanh, maxpool).

CROWN Backward Bound Propagation

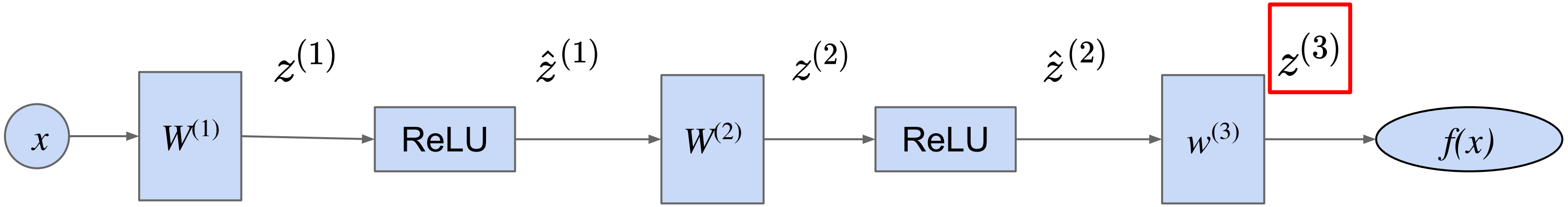
- In CROWN, we **propagate a linear lower bound** for output neuron w.r.t. hidden or input neuron.



- $W^{(1)}$, $W^{(2)}$, $w^{(3)}$ are weights of the NN (output dimension is 1 so $w^{(3)}$ is an vector)
- **Goal**: get a lower bound for $f(x), x \in \mathcal{C}$

CROWN Backward Bound Propagation

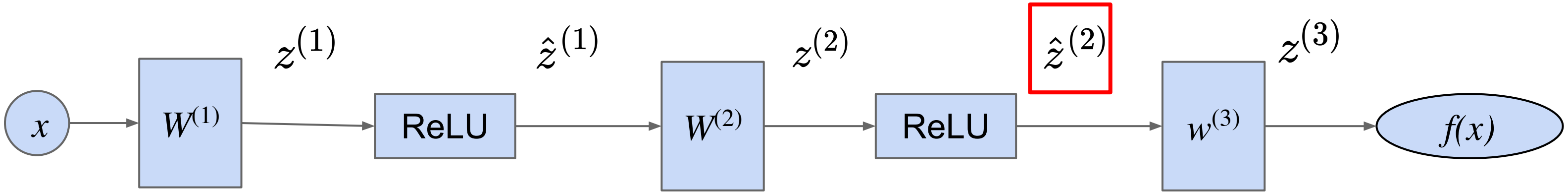
- Goal: find linear relationships between output and every hidden neuron



$$f(x) = z^{(3)}$$

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron

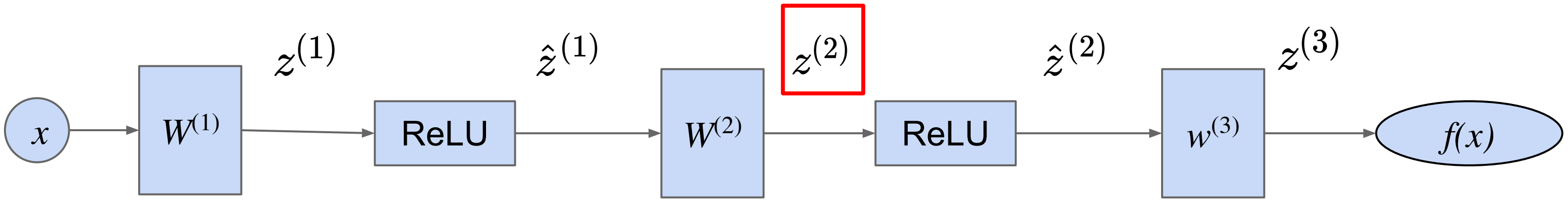


$$f(x) = w^{(3)\top} \hat{z}^{(2)}$$

(By definition)

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron



$$f(x) \geq w^{(3)\top} D^{(2)} z^{(2)} + \text{const.}$$

Encountered an nonlinear operation, need to maintain this inequality.

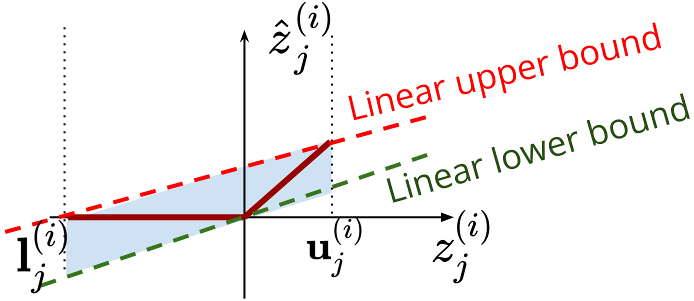
A diagonal matrix $D^{(2)}$ reflects the relaxation of ReLU neurons will be used.

Relaxation During Bound Propagation

- How to design $D^{(2)}$ so the lower and upper bounds are maintained?
- **First step**: for **each unstable** ReLU neuron, linearly lower and upper bound the non-linear function

pre-activation bounds

$$\mathbf{l}_j^{(i)} \leq z_j^{(i)} \leq \mathbf{u}_j^{(i)}$$



$$\boxed{\underline{a}_j^{(i)} z_j^{(i)} + \underline{b}_j^{(i)}} \leq \hat{z}_j^{(i)} := \mathrm{ReLU}(z_j^{(i)}) \leq \boxed{\overline{a}_j^{(i)} z_j^{(i)} + \overline{b}_j^{(i)}}$$

Relaxation During Bound Propagation

- **Second step**: Take the lower or upper bound based on the worst-case

lower bound each term!

Goal: lower bound $f(x) := w^{(3)\top}\mathrm{ReLU}(z^{(2)}) := w^{(3)\top}\hat{z}^{(2)} = \sum_j \boxed{w_j^{(3)} \cdot \hat{z}_j^{(2)}}$

- Take the lower bound of $\hat{z}_j^{(2)}$ when $w_j^{(3)}$ is positive
- Take the upper bound of $\hat{z}_j^{(2)}$ when $w_j^{(3)}$ is negative

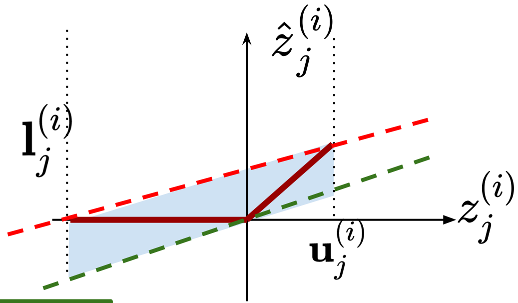
$$\sum_j w_j^{(3)} \cdot \hat{z}_j^{(2)} \geq \sum_{j, w_j^{(3)} \geq 0} w_j^{(3)} \cdot \text{lower bound of } \hat{z}_j^{(2)} + \sum_{j, w_j^{(3)} < 0} w_j^{(3)} \cdot \text{upper bound of } \hat{z}_j^{(2)}$$

Relaxation During Bound Propagation

- **Second step**: Take the lower or upper bound based on the worst-case

Goal: lower bound $\sum_j w_j^{(3)} \cdot \hat{z}_j^{(2)} \ge \sum_{j, w_j^{(3)} \ge 0} w_j^{(3)} \cdot \boxed{\text{lower bound of } \hat{z}_j^{(2)}} + \sum_{j, w_j^{(3)} < 0} w_j^{(3)} \cdot \boxed{\text{upper bound of } \hat{z}_j^{(2)}}$

replace with linear bounds



$$\boxed{\underline{a}_j^{(i)} z_j^{(i)} + \underline{b}_j^{(i)}} \le \hat{z}_j^{(i)} := \mathrm{ReLU}(z_j^{(i)}) \le \boxed{\overline{a}_j^{(i)} z_j^{(i)} + \overline{b}_j^{(i)}}$$

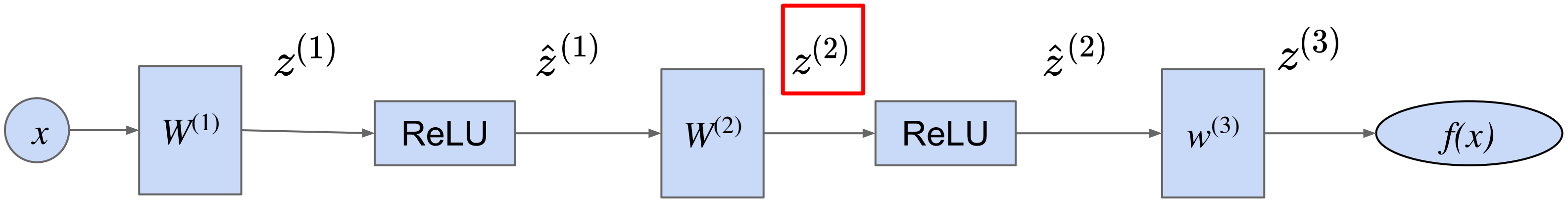
Rearrange (ignore bias terms):

$$w^{(3)\top} \hat{z}^{(2)} \ge w^{(3)\top} D^{(2)} z^{(2)} + \text{bias}$$

Diagonal matrix $D_{j,j}^{(2)} = \begin{cases} \underline{a}_j^{(2)}, & w_j^{(3)} \ge 0 \\ \overline{a}_j^{(2)}, & w_j^{(3)} < 0 \end{cases}$

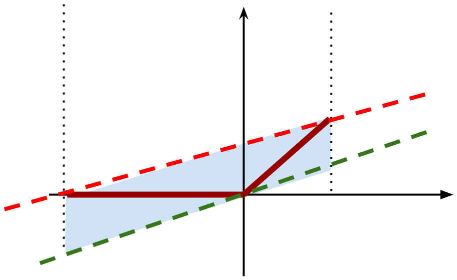
CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron



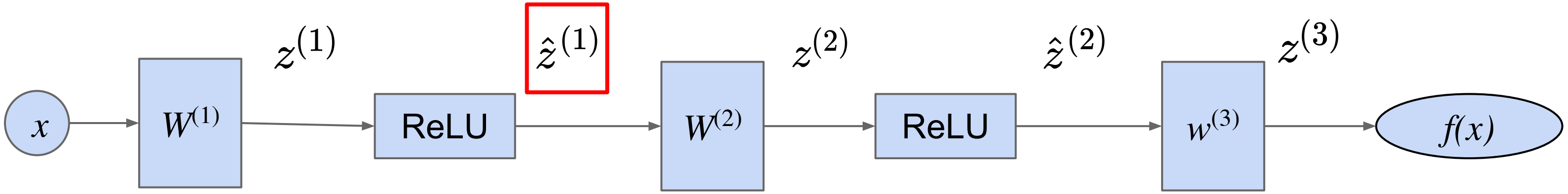
$$f(x) \geq w^{(3)\top} D^{(2)} z^{(2)} + \text{const.}$$

$D^{(2)}$ depends on the signs in $w^{(3)}$, and the linear relaxation of ReLU neuron to make the inequality hold



CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron



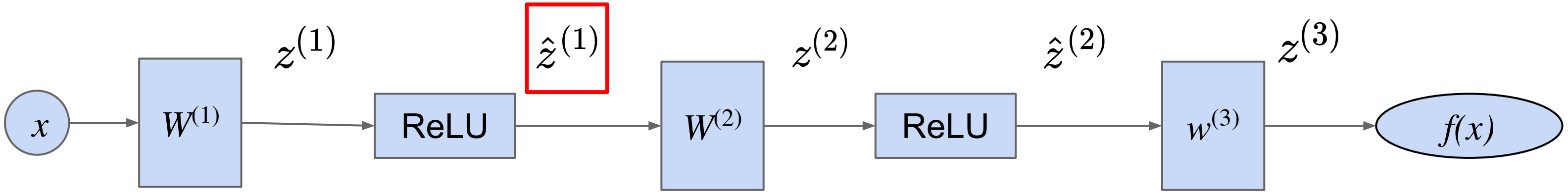
$$f(x) \geq w^{(3)\top} D^{(2)} \boxed{z^{(2)}} + \text{const.}$$

By definition $z^{(2)} = W^{(2)} \hat{z}^{(1)}$

The rest layers follow the same way of propagating bounds

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron

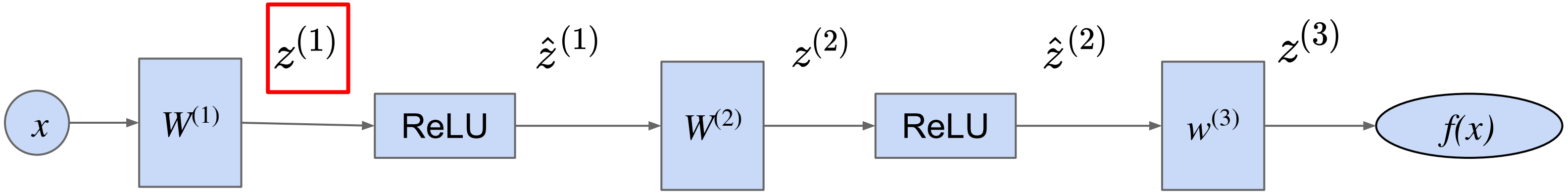


$$f(x) \geq w^{(3)\top} D^{(2)} W^{(2)} \hat{z}^{(1)} + \text{const.}$$

The rest layers follow the same way of propagating bounds

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron



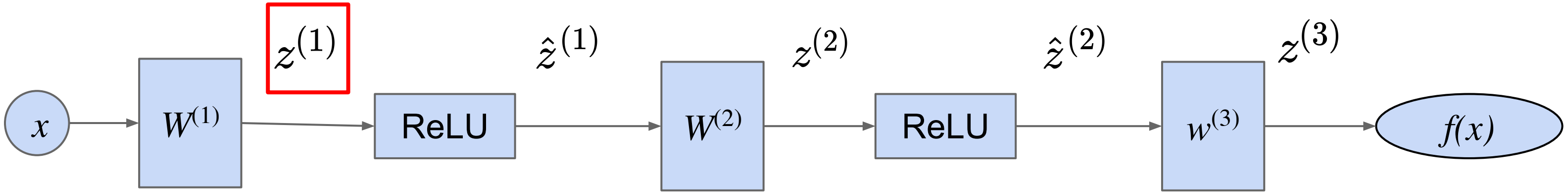
$$f(x) \geq w^{(3)\top} D^{(2)} W^{(2)} \boxed{D^{(1)}} z^{(1)} + \text{const.}$$

Based on the linear relaxation of ReLU

The rest layers follow the same way of propagating bounds

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron



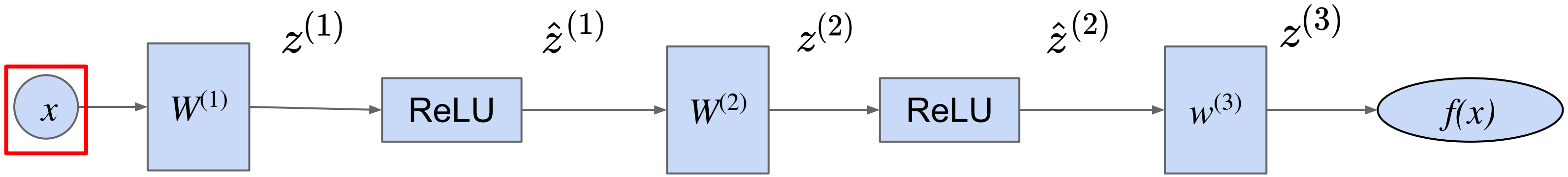
$$f(x) \geq w^{(3)\top} D^{(2)} W^{(2)} D^{(1)} \boxed{z^{(1)}} + \text{const.}$$

By definition $z^{(1)} = W^{(1)} x$

The rest layers follow the same way of propagating bounds

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron, until we reach the input!

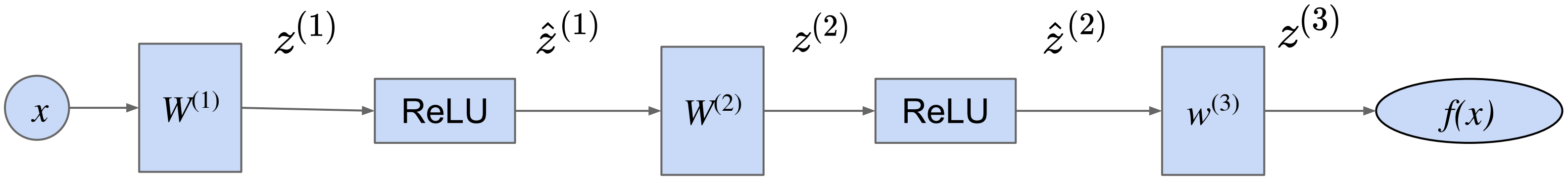


$$f(x) \geq w^{(3)\top} D^{(2)} W^{(2)} D^{(1)} W^{(1)} x + \text{const.}$$

The rest layers follow the same way of propagating bounds

CROWN Backward Bound Propagation

- Goal: find linear relationships between output and every hidden neuron, until we reach the input!



$$f(x) \geq \boxed{w^{(3)\top} D^{(2)} W^{(2)} D^{(1)} W^{(1)}} x + \text{const.}$$

CROWN linear bound: $\min_{x \in \mathcal{C}} f(x) \geq \min_{x \in \mathcal{C}} \boldsymbol{a}_{\text{CROWN}}^{\top} x + c_{\text{CROWN}} := \min_{x \in \mathcal{C}} f_{\text{CROWN}}(x)$

Where $\boldsymbol{a}_{\text{CROWN}}$ and $\boldsymbol{c}_{\text{CROWN}}$ are functions of NN weights, and can be computed efficiently on GPUs in a backward manner

The CROWN Lower Bound

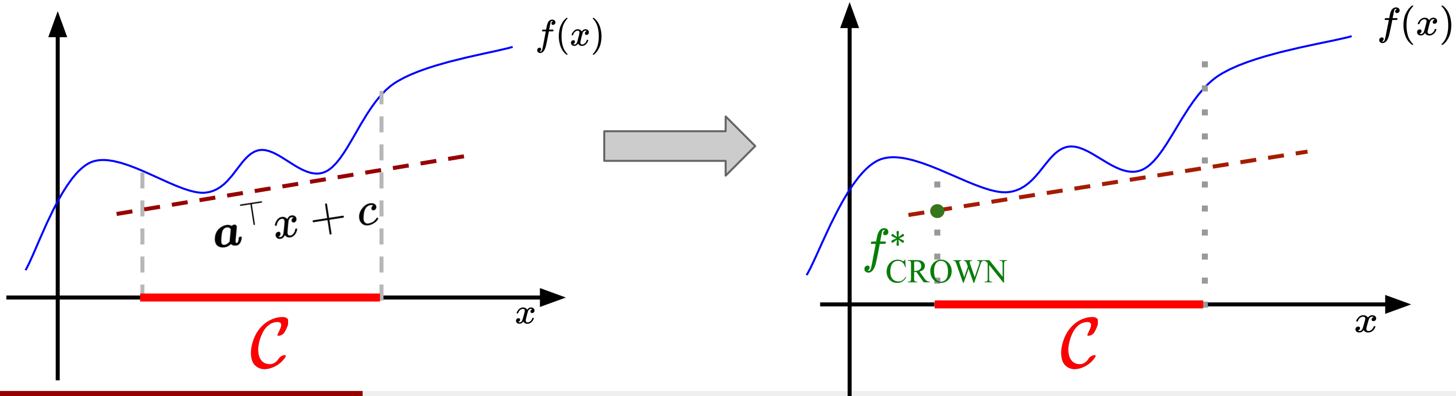
Linear Bound: $$f_{\text{CROWN}}(x) = \boldsymbol{a}_{\text{CROWN}}^\top x + c_{\text{CROWN}}$$

Final lower bound by solving an easier linear optimization problem:

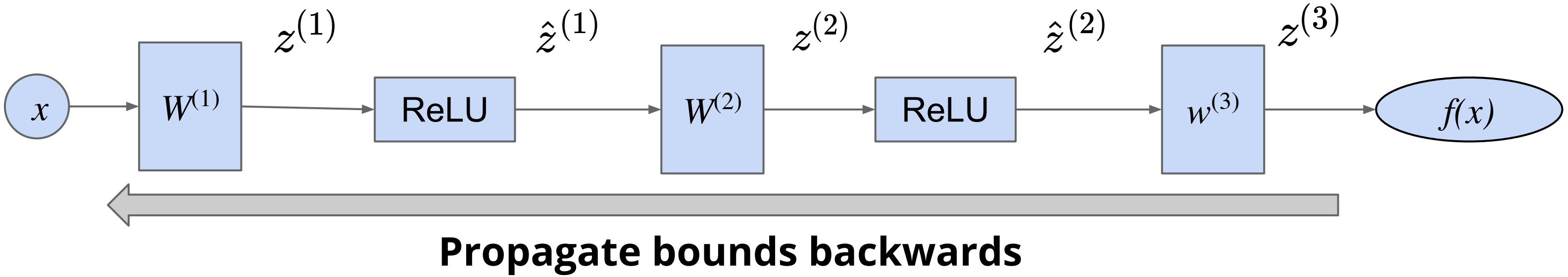
$$f^*_{\text{CROWN}} = \min_{x \in \mathcal{C}} \boldsymbol{a}_{\text{CROWN}}^\top x + c_{\text{CROWN}}$$

Simple closed form for ℓ_∞ norm perturbation $x \in \{x \mid \|x - x_0\|_\infty \leq \epsilon\}$

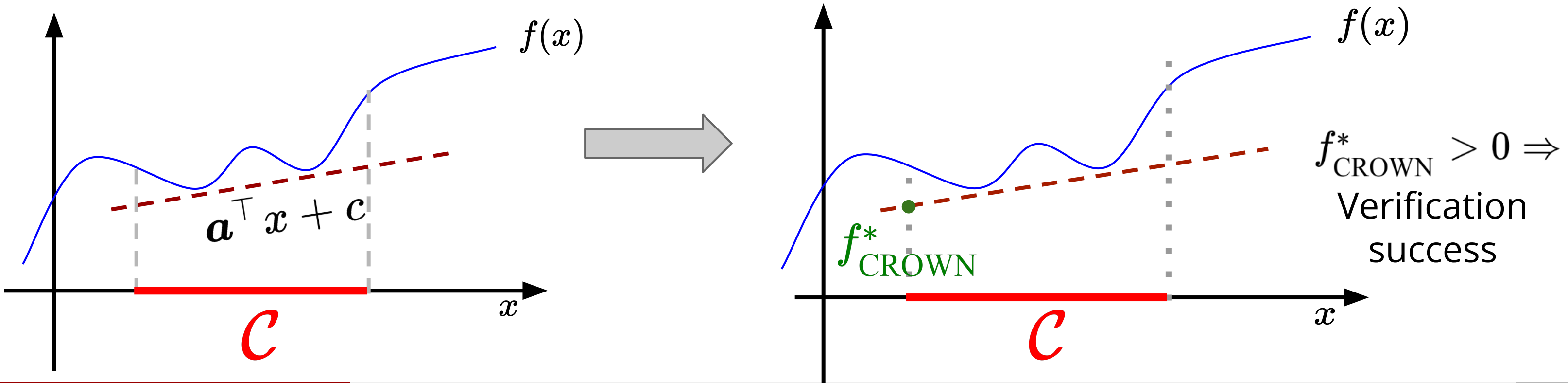
$$f^*_{\text{CROWN}} = -\|\boldsymbol{a}_{\text{CROWN}}\|_1 \epsilon + \boldsymbol{a}_{\text{CROWN}}^\top x_0 + c_{\text{CROWN}}$$



CROWN Backward Bound Propagation: Summary

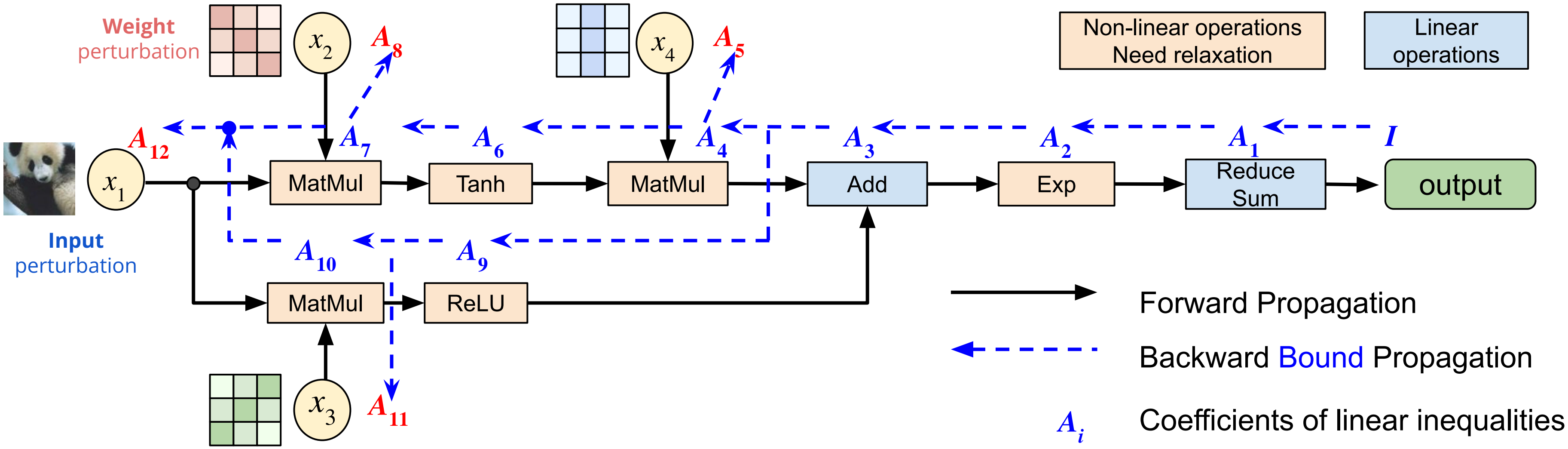


$$\min_{x\in\mathcal{C}} f(x) \geq \min_{x\in\mathcal{C}} \boldsymbol{a}_{\text{CROWN}}^{\top} x + c_{\text{CROWN}} := \min_{x\in\mathcal{C}} f_{\text{CROWN}}(x)$$



Bound Propagation for Any Computation

We can generalize CROWN to a **graph algorithm** to run on **general** computational graphs (a superset of many existing algorithms on verifying the robustness of LSTMs, Transformers, etc)

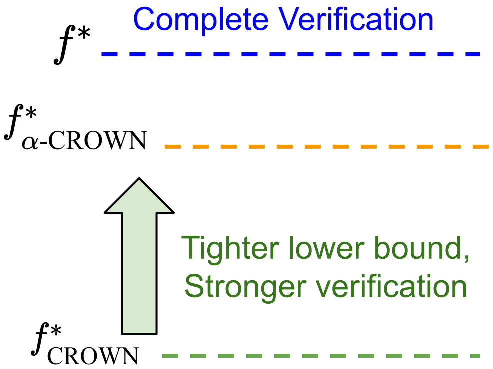


"Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond". Kaidi Xu*, Zhouxing Shi*, Huan Zhang*, Yihan Wang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, Cho-Jui Hsieh. NeurIPS 2020.

Outline: Algorithms for Neural Network Verification

CROWN:
bound propagation
based verification
(Zhang et al. NIPS 2018)

α-CROWN:
tighten bounds with
optimization on GPU
(Xu et al. ICLR 2021)



[1] Efficient Neural Network Robustness Certification with General Activation Functions, **Huan Zhang***, Tsui-Wei Weng*, Pin-Yu Chen, Cho-Jui Hsieh, Luca Daniel. NIPS 2018.

[2] Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. Kaidi Xu*, Zhouxing Shi*, **Huan Zhang***, Yihan Wang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, Cho-Jui Hsieh. NeurIPS 2020.

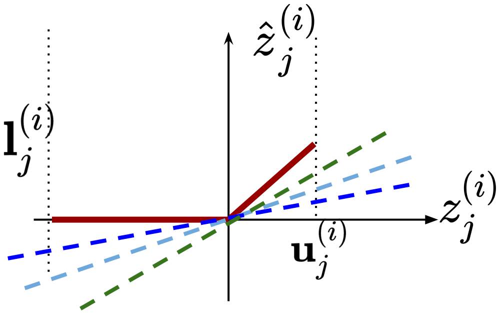
[3] Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers, Kaidi Xu*, **Huan Zhang***, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, Cho-Jui Hsieh. **ICLR 2021**.

[4] Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. Shiqi Wang*, **Huan Zhang***, Kaidi Xu*, Xue Lin, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter. NeurIPS 2021.

(***Co-first authors**).

α-CROWN: Further Tighten the Bounds

- ReLU neurons have a flexible lower bound for relaxation
- Try different lower bounds to find the tightest bound
- Each unstable ReLU has a lower bound to select, so lots of freedom here



$$\hat{z}_j^{(i)} \geq \boldsymbol{\alpha}_j^{(i)} z_j^{(i)} \quad \boxed{(0 \leq \boldsymbol{\alpha}_j^{(i)} \leq 1)}$$

Adjustable!

$\mathbf{l}_j^{(i)} \leq z_j^{(i)} \leq \mathbf{u}_j^{(i)}$ are pre-activation bounds, also computed using CROWN

α-CROWN: Bound Propagation with Optimized Bounds

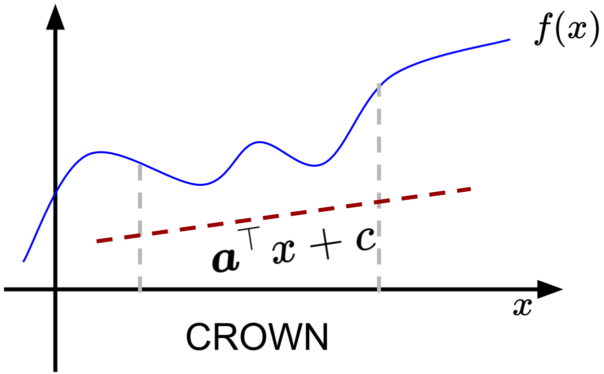
$$f^*_{\text{CROWN}} = \min_{x \in \mathcal{C}} \boldsymbol{a}_{\text{CROWN}}^\top x + c_{\text{CROWN}}$$

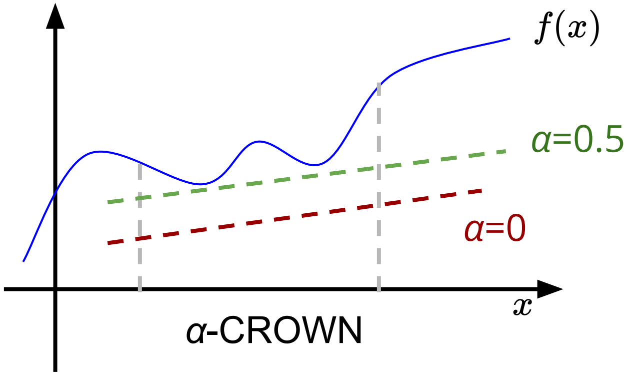
Actually a function of $\boldsymbol{\alpha}$. How to effectively optimize $\boldsymbol{\alpha}$ to find the best bound?

Key idea: tighten bounds using gradients

$$f^* = \min_{x \in \mathcal{C}} f(x) \geq \boxed{\max_{0 \leq \boldsymbol{\alpha} \leq 1}} \min_{x \in \mathcal{C}} f_{\text{CROWN}}(x; \boldsymbol{\alpha})$$

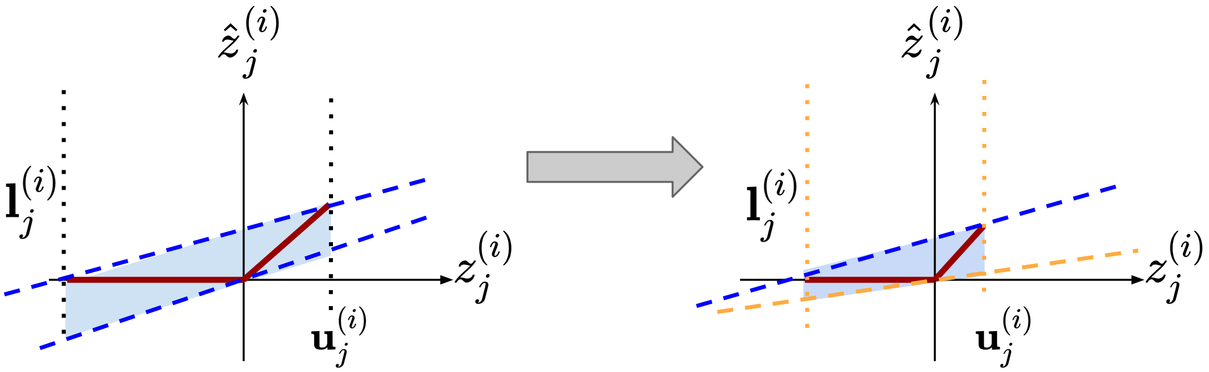
Inner minimization can usually be solved in closed form

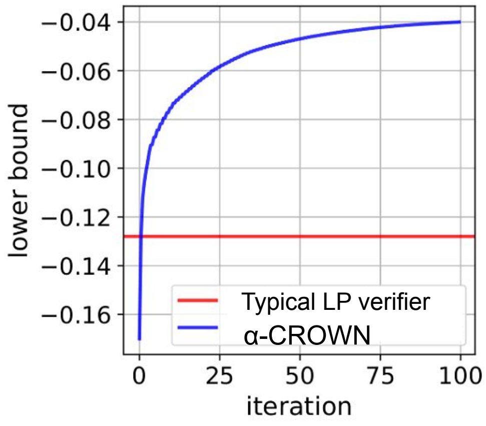




α-CROWN: Summary

- We can use gradient to optimize the relaxation, to make the bound tighter (tighter bound => stronger incomplete verification)
- We can make the bound **tighter** than the more expensive LP-based verifiers
- Optimization can be done rapidly on **GPUs**





Outline: Algorithms for Neural Network Verification

CROWN:
bound propagation
based verification
(Zhang et al. NIPS 2018)

→

α-CROWN:
tighten bounds with
optimization on GPU
(Xu et al. ICLR 2021)

→

β-CROWN:
bound propagation +
branch and bound
(Wang et al. NeurIPS 2021)

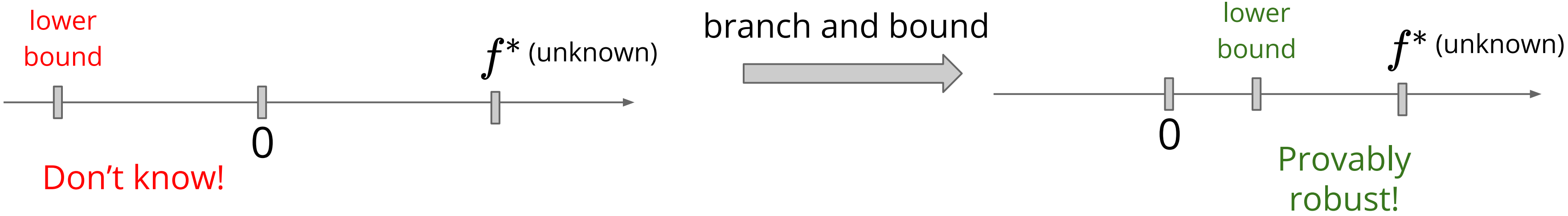
[1] Efficient Neural Network Robustness Certification with General Activation Functions, Huan Zhang*, Tsui-Wei Weng*, Pin-Yu Chen, Cho-Jui Hsieh, Luca Daniel. NIPS 2018.
[2] Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. Kaidi Xu*, Zhouxing Shi*, Huan Zhang*, Yihan Wang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, Cho-Jui Hsieh. NeurIPS 2020.
[3] Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers, Kaidi Xu*, Huan Zhang*, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, Cho-Jui Hsieh. ICLR 2021.
[4] Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. Shiqi Wang*, **Huan Zhang***, Kaidi Xu*, Xue Lin, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter. **NeurIPS 2021**.
(*Co-first authors).

Review of Complete and Incomplete Verification Problem

We aim to solve:

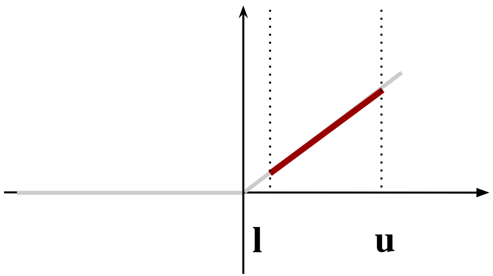
$$f^* = \min_{x \in \mathcal{C}} f(x)$$

- CROWN (bound propagation methods) is usually too weak to verify many practical models (lower bound too loose)
- We use branch and bound to improve the bounds, and can converge to f^*

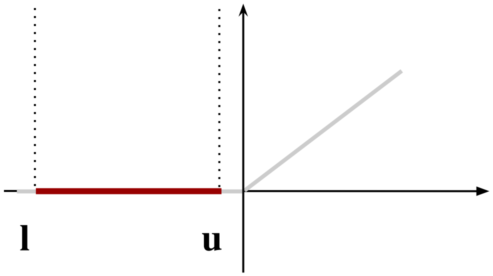


Branch and Bound for ReLU Network Verification

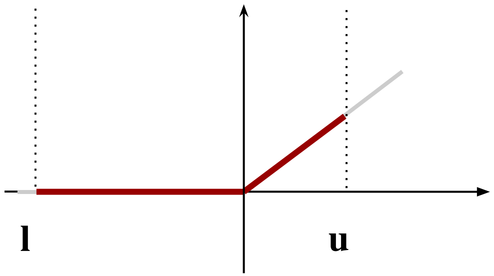
Recall that ReLU neurons have three cases depending on pre-activation bounds:



$\mathbf{l} \geq 0$, always active (linear)

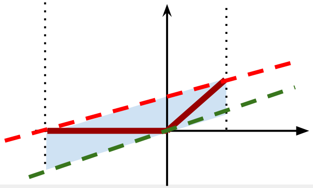


$\mathbf{u} \leq 0$, Always inactive (zero)



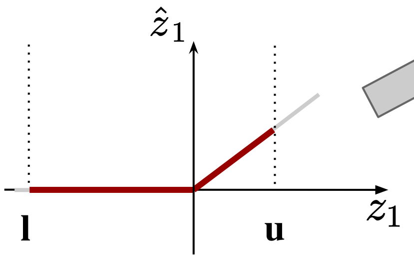
$\mathbf{l} \leq 0 \leq \mathbf{u}$

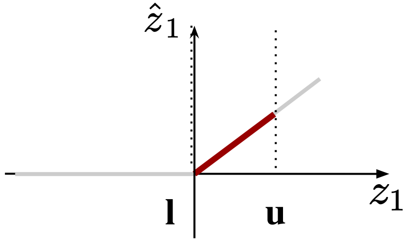
Unstable (non-linear)
Must be **relaxed**; relaxation makes bounds looser



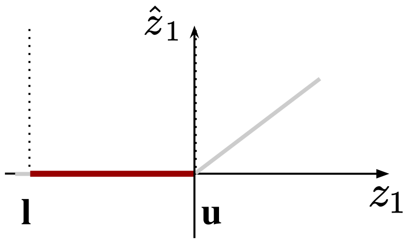
Branch and Bound: The Branching Step

Split each "unstable" ReLU neurons to two subproblems:





Relu is linear, no relaxation needed



Additional linear constraint ("split constraint"):

$$z_1 > 0$$

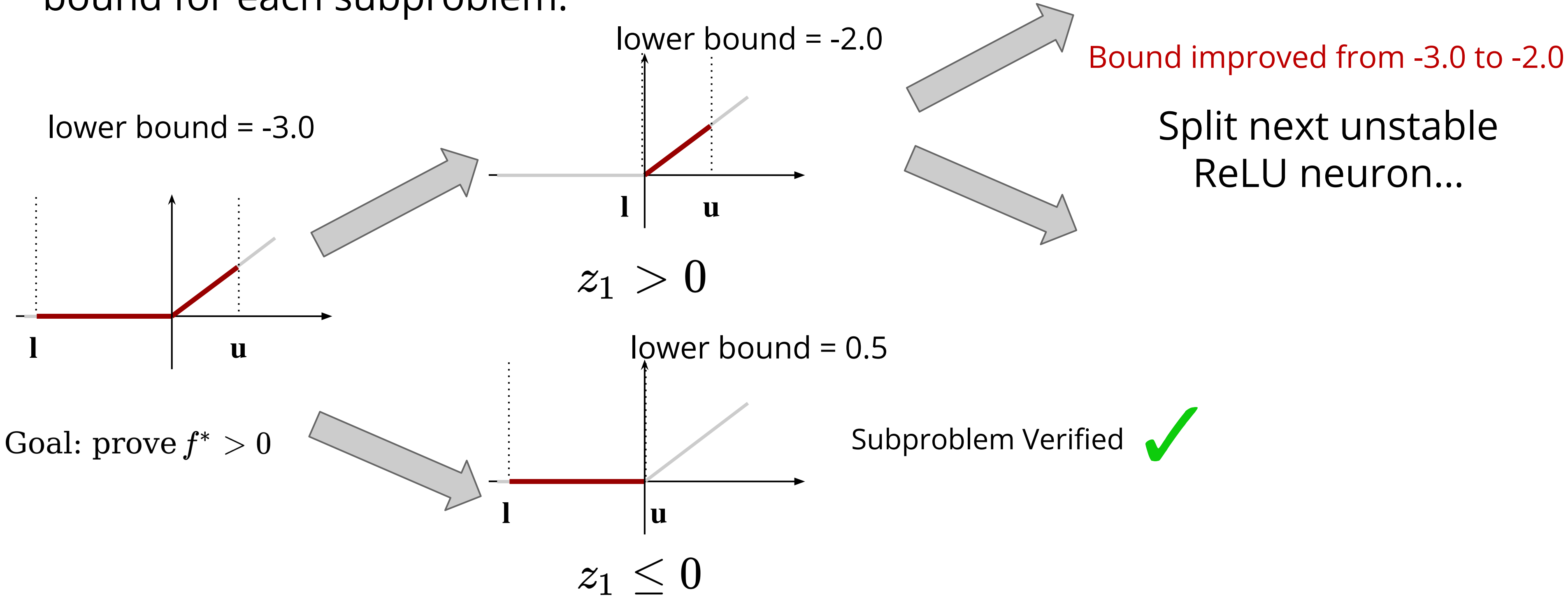
OR

$$z_1 \leq 0$$

The additional constraint can make bounds tighter

Branch and Bound: The Bounding Step

Using an incomplete solver (traditionally, LP-based verifier) to get the lower bound for each subproblem:



Branch and Bound Search Tree

Branch and bound improves the lower bound

Lower bound = -3.0
(computed by a incomplete verifier)

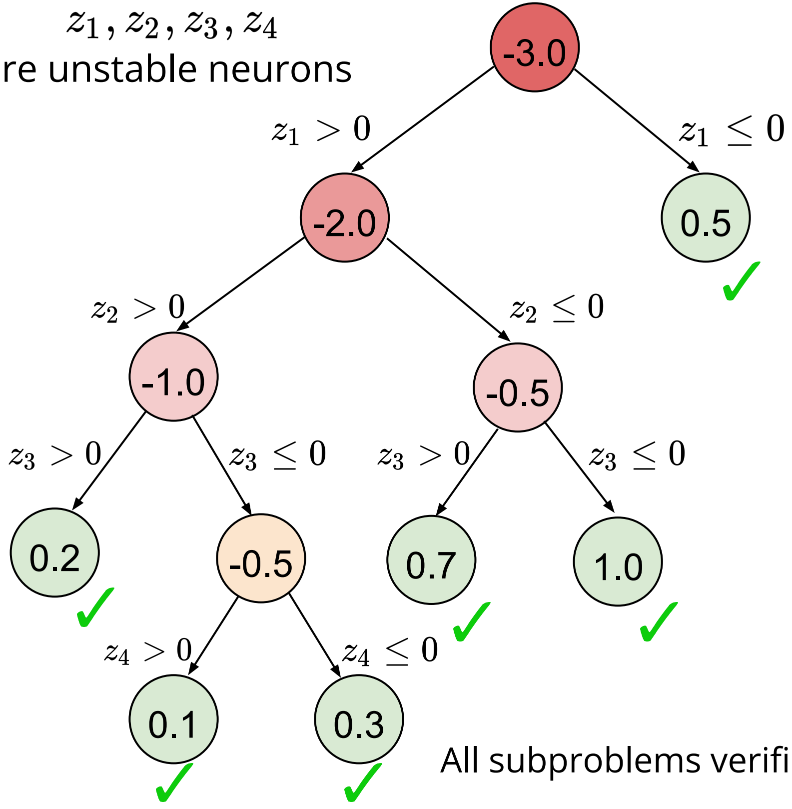
Lower bound = min(-2.0, 0.5) = -2.0

Lower bound = min(-1.0, -0.5) = -1.0

Lower bound = -0.5

Lower bound = 0.1

z_1, z_2, z_3, z_4
are unstable neurons

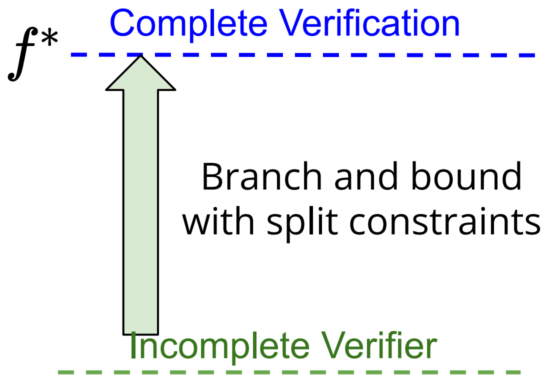


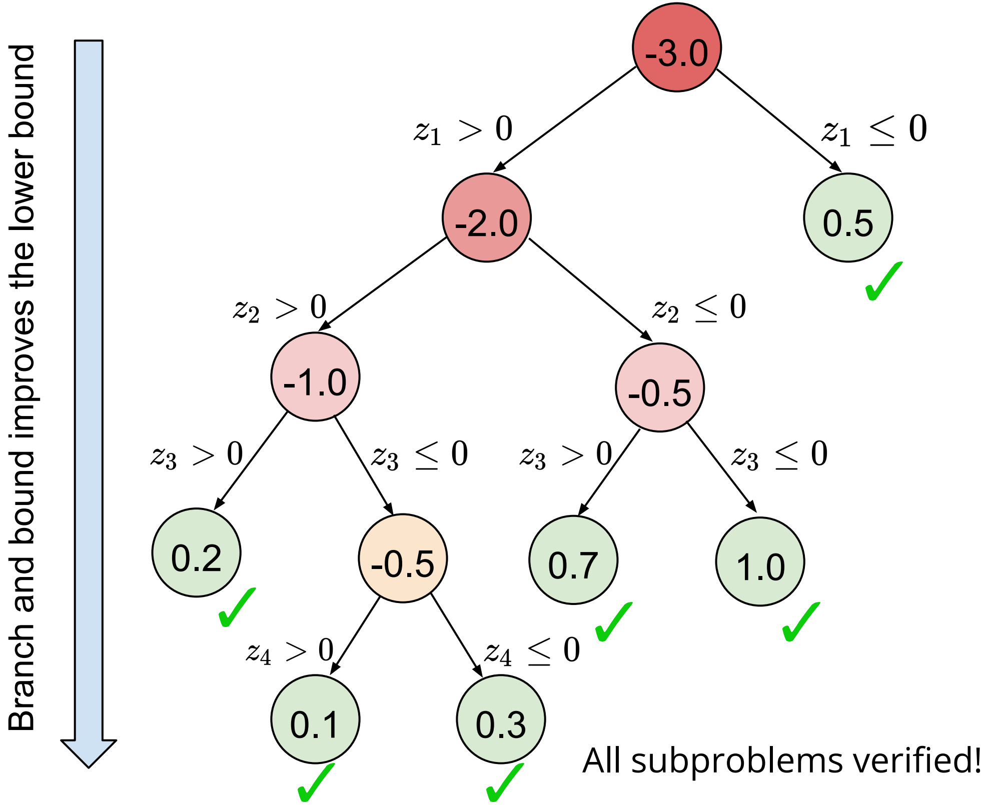
All subproblems verified!

Branch and Bound Search Tree

Branch and bound is complete if each relaxed subproblem (**with split constraint**) can be solved to optimal.

$$f^* = \min_{x \in \mathcal{C}} f(x)$$





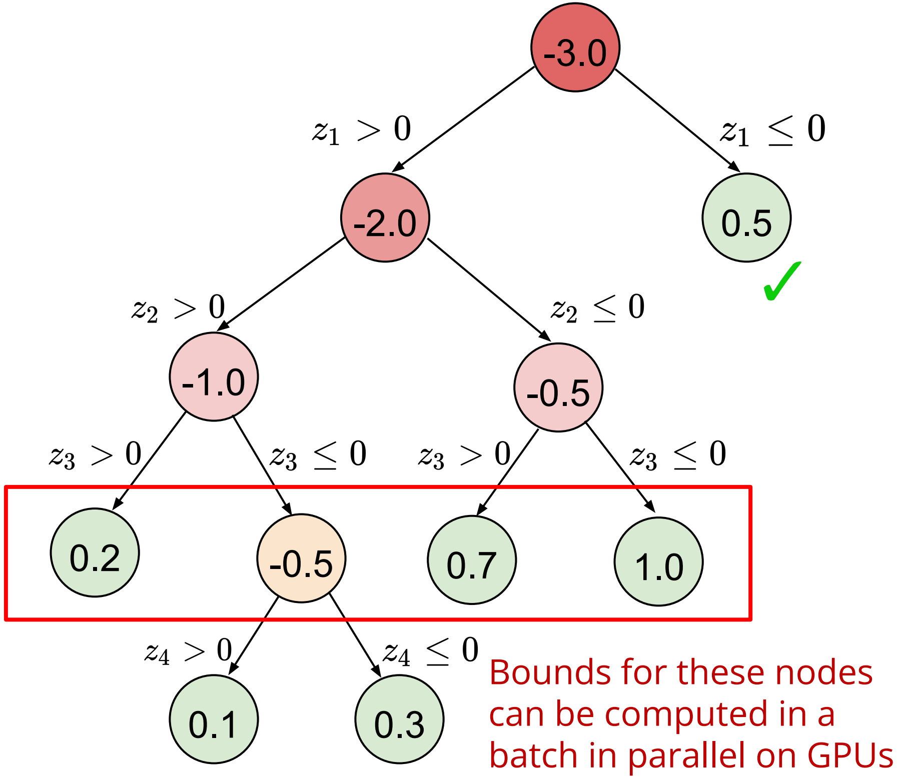
Branch and Bound with GPU Parallelization

Idea:

Combine rapid bound propagation based incomplete verifiers on GPUs with branch and bound (BaB) to achieve complete verification

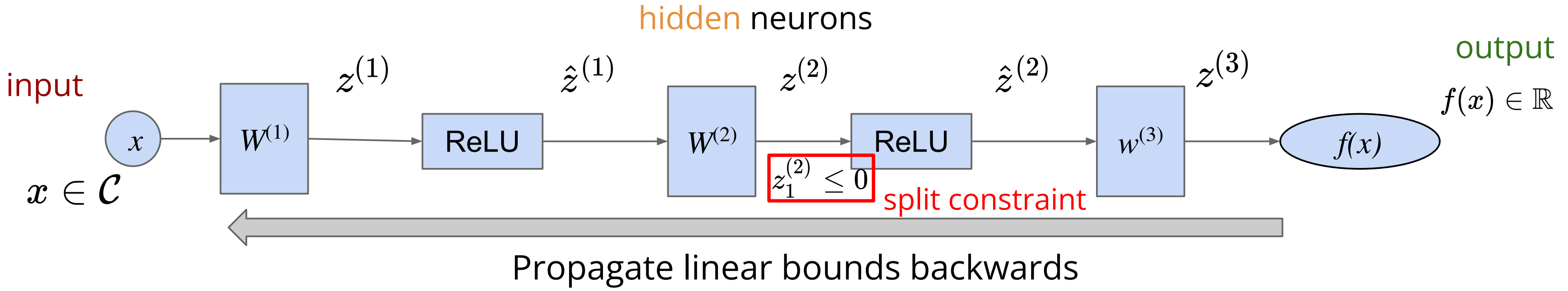
Outcome:

up to 100-1000x faster than MIP based approach, enable us to scale complete verification to larger models



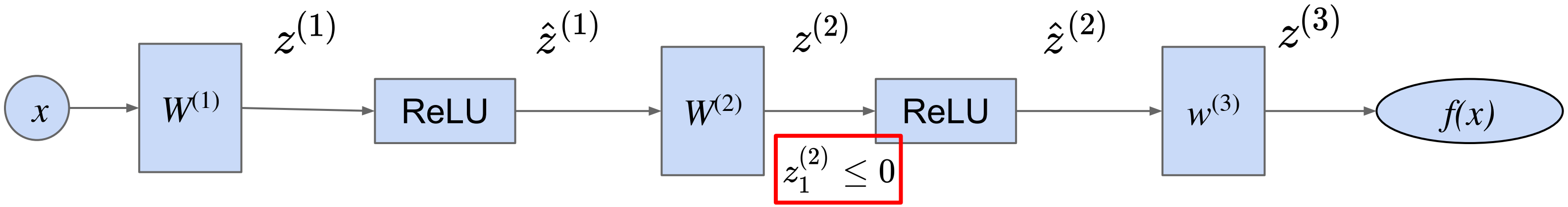
β-CROWN: Bound Propagation with Split Constraints

- To use branch and bound, bound propagation must incorporate the split constraints; CROWN *cannot* handle it



β-CROWN: Bound Propagation with Split Constraints

- Deal with split constraints with Lagrangians



CROWN: $$\min_{x\in\mathcal{C}} f(x) \ge \min_{x\in\mathcal{C}} w^{(3)\top} D^{(2)} z^{(2)} + \text{const.}$$ Cannot handle split constraint

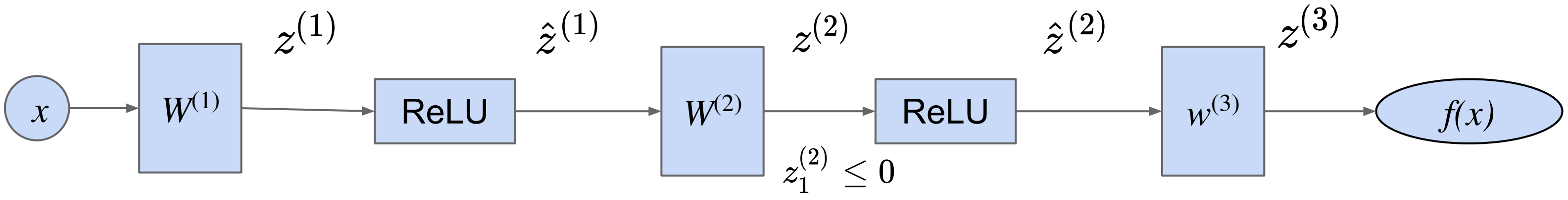
***β*-CROWN**: $$\min_{x\in\mathcal{C}, \boxed{z_1^{(2)}<0}} f(x) \ge \max_{\beta\ge 0}\min_{x\in\mathcal{C}} w^{(3)\top} D^{(2)} z^{(2)} + \boxed{\beta^\top S^{(2)} z^{(2)}} + \text{const.}$$

Lagrangian/KKT multipliers

S is an diagonal matrix with +/-1 and 0

β-CROWN: Bound Propagation with Split Constraints

- Lagrangians are also propagated!



CROWN: $$\min_{x\in\mathcal{C}} f(x) \geq \min_{x\in\mathcal{C}} \boxed{w^{(3)\top} D^{(2)}} z^{(2)} + \text{const.}$$

***β*-CROWN**: $$\min_{x\in\mathcal{C}, z_1^{(2)}\in<0} f(x) \geq \max_{\beta\geq 0} \min_{x\in\mathcal{C}} \boxed{\left(w^{(3)\top} D^{(2)} + \beta^\top S^{(2)}\right)} z^{(2)} + \text{const.}$$

Linear coefficients changed with one additional term during propagation

β-CROWN: Bound Propagation with Split Constraints

β-CROWN main theorem:

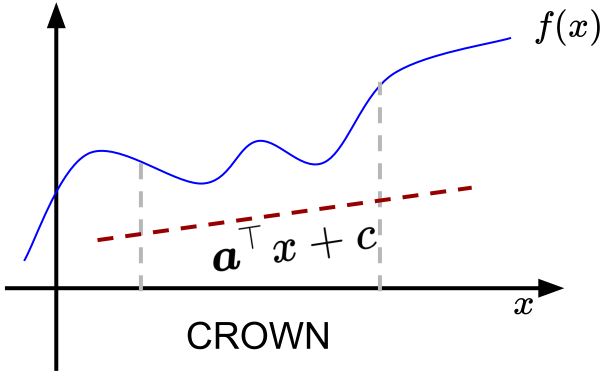
$$\min_{\underset{\text{all split constraints}}{x\in\mathcal{C},z\in\mathcal{Z}}} f(x) \geq \max_{\boldsymbol{\beta}\geq 0} \min_{x\in\mathcal{C}} (\boldsymbol{a}+\mathbf{P}\boldsymbol{\beta})^\top x + \mathbf{q}^\top\boldsymbol{\beta} + c$$

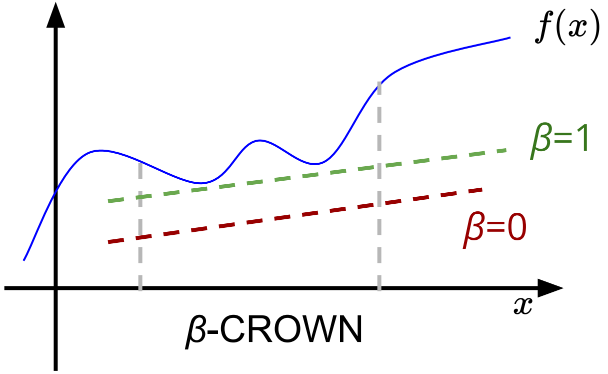
all split constraints

Compared to (vanilla) CROWN (β=0):

$$\min_{x\in\mathcal{C}} f(x) \geq \min_{x\in\mathcal{C}} \boldsymbol{a}^\top x + c$$

Different $\boldsymbol{\beta}$ corresponds to different bounds, and we can choose the tightest one





Next Part

`auto_LiRPA`: Easy incomplete verification with PyTorch!

***α,β*-CROWN**: Award-winning complete verifier with SOTA performance