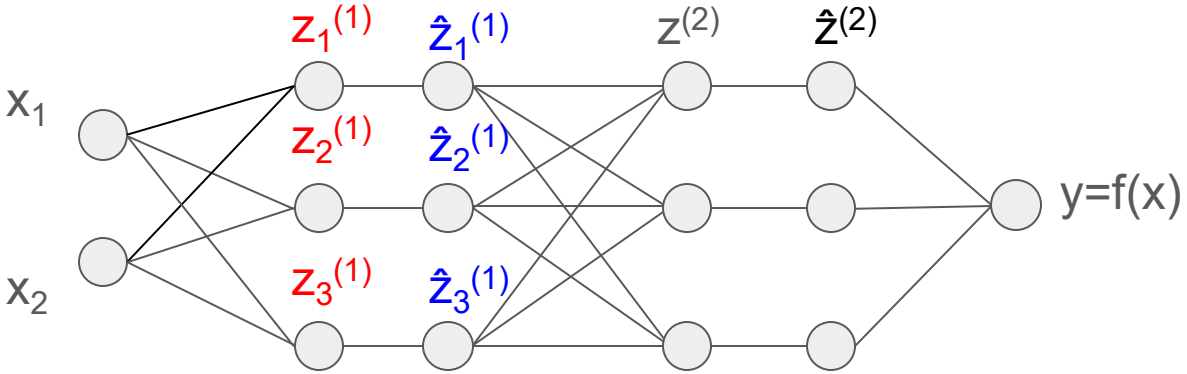
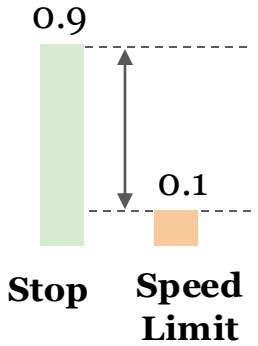
Neural Network Verification
Integer and Linear Programming Formulations

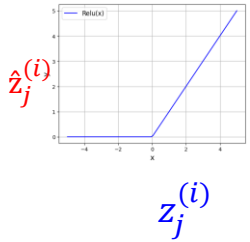
Prof. Sayan Mitra

mitras@Illinois.edu

Review: Neural network verification as a satisfiability problem







To verify for all $x \in S$, $y > 0 \wedge y = f(x)$ we solve

Does there exist x, s.t. $x \in S \wedge y \leq 0 \wedge y = f(x)$

Input domain under consideration

Negation of the desired property

Defines the neural network

Verification of neural networks

Satisfiability problem: $\exists\ x \in S\ \wedge\ y \leq 0\ \wedge\ y = f(x)$

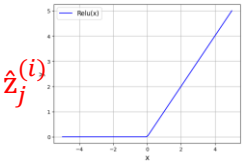
$x_i \leq u_i\ \wedge x_i \geq l_i$ for each dimension of x

$((z_j^{(i)} \geq 0 \wedge \hat{z}_j^{(i)} = z_j^{(i)})\ \vee\ (z_j^{(i)} < 0 \wedge \hat{z}_j^{(i)} = 0))$ for each ReLU neuron

$z_1 = W^{(1)} x \quad \wedge \quad z^{(2)} = W^{(2)} \hat{z}^{(1)} \quad \wedge \quad y = w^{(3)T} \hat{z}^{(2)} \quad \wedge \quad y \leq 0$

Add all clauses to the formula and solve using DPLL(T) with Linear Real Arithmetic.

In general this is very slow!



Reluplex: Decision Procedure for ReLU NNs

Input: F in Reluplex form

Output: $\exists \boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

- Delay case splitting on ReLUs
- In the worst case it is still exponential, but has been shown to be empirically better than DPLL^{LRA}

Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks
Guy Katz, Clark Barrett, David Dill, Kyle Julian, Mykel Kochenderfer, 2017

Reluplex Decision Procedure for ReLU NNs

Input: F in Reluplex form

Output: $\exists \boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

Reluplex form (Recall Simplex form)

- Equations (same as Simplex)
- Bounds (same as Simplex)
- Relu: $x_i = relu(x_j)$

Given conjunction of inequalities and ReLU constraints we can convert them to Reluplex form and add $x_i \geq 0$.

Reluplex

Input: A formula F in Reluplex form

Output: $\boldsymbol{x} \vDash F$ or UNSAT

$\boldsymbol{x} \coloneqq \langle x_i \mapsto 0 \rangle$; F':= non ReLU part of F

while true **do**

$r \coloneqq Simplex(F', \boldsymbol{x})$

if r **is unsat then return UNSAT**

else if $r \vDash F$ **then return** r

// Handle violated ReLU constraints

Let $x_i = relu(x_j) \in F$ such that $\boldsymbol{x}[x_i \neq relu(\boldsymbol{x}[x_j)$

// Case split

If $u_j > 0, l_j < 0$ and $x_i = relu(x_j)$ considered $> \tau$ times

Reluplex

Input: A formula F in Reluplex form

Output: $\boldsymbol{x} \vDash F$ or UNSAT

$\boldsymbol{x} \coloneqq \langle x_i \mapsto 0 \rangle$; F':= non ReLU part of F

while true do

$r \coloneqq Simplex(F', \boldsymbol{x})$

if r is unsat then return UNSAT

else if $r \vDash F$ then return r

// Handle violated ReLU constraints

Let $x_i = relu(x_j) \in F$ such that $\boldsymbol{x}.x_i \neq relu(\boldsymbol{x}.x_j)$

if x_i is basic then pivot x_i with non-basic variable x_k where $k \neq j$ and $c_{ik} \neq 0$

if x_j is basic then pivot x_j with non-basic variable x_k where $k \neq i$ and $c_{jk} \neq 0$

$\boldsymbol{x}.x_i \coloneqq relu(\boldsymbol{x}.x_j)$ OR $\boldsymbol{x}.x_j \coloneqq \boldsymbol{x}.x_i$

// Case split

If $u_j > 0, l_j < 0$ and $x_i = relu(x_j)$ considered > τ times

$r_1 = Reluplex(F \wedge x_j \geq 0 \wedge x_j = x_j)$

$r_2 = Reluplex(F \wedge x_j \leq 0 \wedge x_i = 0)$

if $r_1 = r_2 =$ unsat then return UNSAT

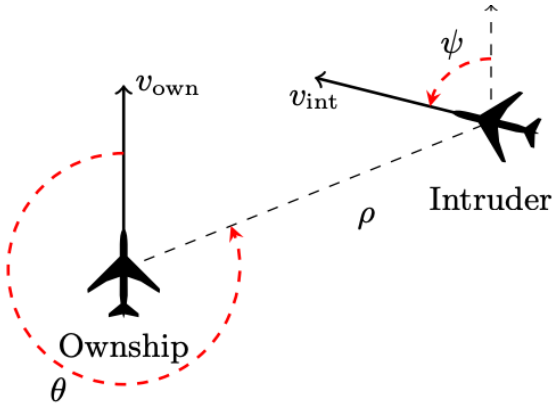
else if $r_1 = sat$ then return r_1 else return r_2

Termination

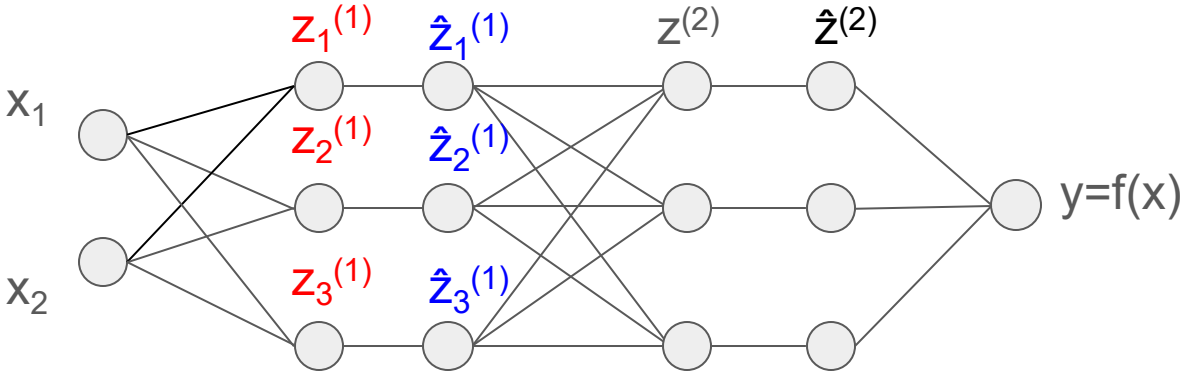
- The last part of the algorithm ensures termination
- Otherwise, the algorithm could loop forever between fixing relu constraints and Simplex
- $F \equiv x_i = relu(x_j)$ is split into two cases
  - $F_1 \equiv \ x_j \geq 0 \wedge x_i = x_j$
  - $F_2 \equiv \ x_j \leq 0 \wedge x_i = 0$
  - $F' \equiv (F \wedge F_1) \vee (F \wedge F_2)$

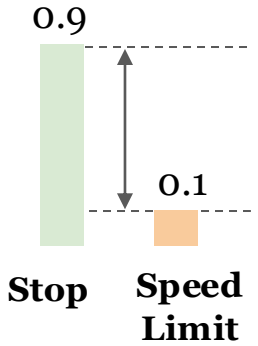
Performance of Reluplex on ACAS

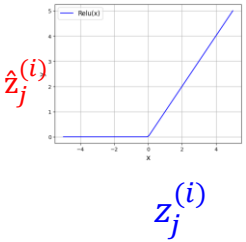
	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9



Review: Neural network verification as a satisfiability problem







Does there exist x, s.t. $x \in S \wedge y \leq 0 \wedge y = f(x)$

Input domain under consideration

Negation of the desired property

Defines the neural network

Verification of neural networks

Satisfiability problem: $\exists\ x \in S\ \wedge\ y \leq 0\ \wedge\ y = f(x)$

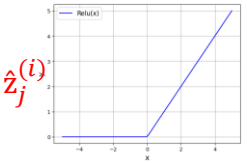
$x_i \leq u_i\ \wedge x_i \geq l_i$ for each dimension of x

$((z_j^{(i)} \geq 0 \wedge \hat{z}_j^{(i)} = z_j^{(i)})\ \vee\ (z_j^{(i)} < 0 \wedge \hat{z}_j^{(i)} = 0))$ for each ReLU neuron

$z_1 = W^{(1)} x \quad \wedge \quad z^{(2)} = W^{(2)} \hat{z}^{(1)} \quad \wedge \quad y = w^{(3)T} \hat{z}^{(2)} \quad \wedge \quad y \leq 0$

Add all clauses to the formula and solve using DPLL(T) with Linear Real Arithmetic.

In general this is very slow!



How does a SMT solver solve this problem?

Obtain the abstract version of the problem

$x_i \leq u_i \wedge x_i \geq l_i$ (assuming box constraints)

$((z_j^{(i)} \geq 0 \wedge \hat{z}_j^{(i)} = z_j^{(i)}) \vee (z_j^{(i)} < 0 \wedge \hat{z}_j^{(i)} = 0))$ for each ReLU neuron

$z_1 = W^{(1)} x \quad \wedge \quad z^{(2)} = W^{(2)} \hat{z}^{(1)} \quad \wedge \quad y = w^{(3)T} \hat{z}^{(2)} \quad \wedge \quad y \leq 0$

For each ReLU neuron: $((p_j^{(i)} \wedge q_j^{(i)}) \vee (\neg p_j^{(i)} \wedge r_j^{(i)}))$

All other clauses contain only a single literal, and must be set to True

How does a SMT solver solve this problem?

Now convert to CNF form

For each ReLU neuron: $((p_j^{(i)} \wedge q_j^{(i)}) \vee (\neg p_j^{(i)} \wedge r_j^{(i)}))$

Distribution: $((p_j^{(i)} \vee \neg p_j^{(i)}) \wedge (p_j^{(i)} \vee r_j^{(i)}) \wedge (q_j^{(i)} \vee \neg p_j^{(i)}) \wedge (q_j^{(i)} \vee r_j^{(i)}))$

Rewrite: $(p_j^{(i)} \vee r_j^{(i)}) \wedge (\neg p_j^{(i)} \vee q_j^{(i)}) \wedge (q_j^{(i)} \vee r_j^{(i)}))$

Observe that when $p_j^{(i)}$=True, $q_j^{(i)}$ must be true; $p_j^{(i)}$=False, $r_j^{(i)}$ must be true;

So, the clause $q_j^{(i)} \vee r_j^{(i)}$ is redundant

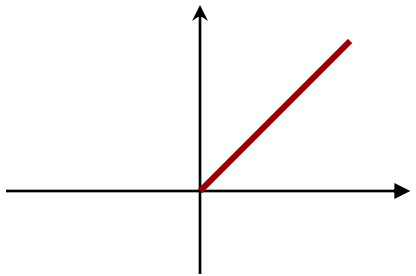
How does a SMT solver solve this problem?

$(p_j^{(i)} \vee r_j^{(i)}) \wedge (\neg p_j^{(i)} \vee q_j^{(i)})$

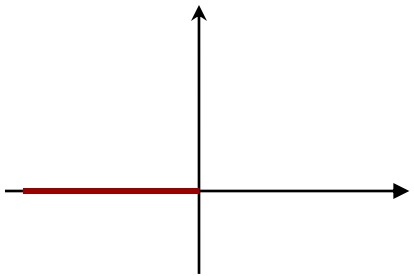
When $p_j^{(i)}$=True, $q_j^{(i)}$ must be true; $p_j^{(i)}$=False, $r_j^{(i)}$ must be true;

SAT solver must try both cases of $p_j^{(i)}$

$p_j^{(i)}$=True $(z_j^{(i)} \geq 0)$



$p_j^{(i)}$=False $(z_j^{(i)} < 0)$



Why using a SMT solver is very slow?

$(p_j^{(i)} \vee r_j^{(i)}) \wedge (\neg p_j^{(i)} \vee q_j^{(i)})$

When $p_j^{(i)}$=True, $q_j^{(i)}$ must be true; $p_j^{(i)}$=False, $r_j^{(i)}$ must be true;

SAT solver must make decisions on $p_j^{(i)}$. In a satisfiable solution from DPLL, each $p_j^{(i)}$ is set to True or False, then a theory solver (Simplex) invoked.

There are exponential number of cases here… and modern neural networks can have millions of neurons!

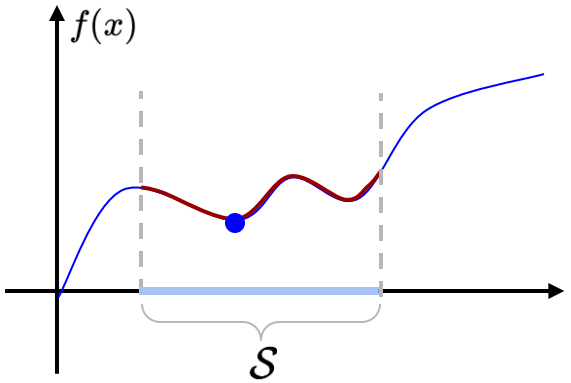
Can we solve the problem without setting every $p_j^{(i)}$?

We will tack this problem from an optimization point of view

Mathematical optimization problems

Given an **objective function** f: S → R

Seek an **optimal solution** x^* such that $f(x^*) \leq f(x)$ for all $x \in S$



Some optimization problems are hard, some are easy

Given an **objective function** f: S → R

Hardness depends on the properties of f and S. For tractable solving they cannot be arbitrary!

- Easy ones: convex optimization, linear programming, semidefinite programming
  - E.g., in linear programming, objective function and constraints must be linear functions

- Hard ones: integer programming, general quadratic programming, general nonlinear programming, …

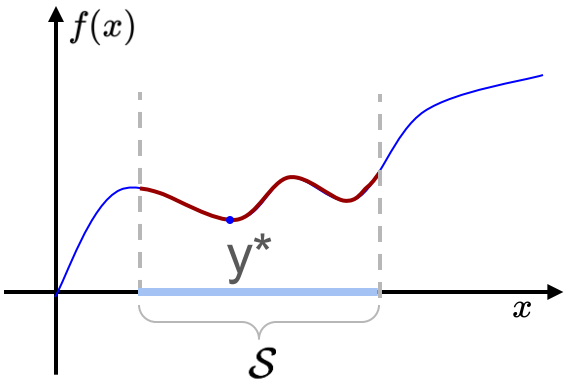
Verification Problem as an optimization problem

$\exists\ x \in S\ \wedge\ y \leq 0\ \wedge\ y = f(x)$

Can be solved with the following **minimization problem:**

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

If the optimal objective y*≤0, then the original Problem is satisfiable



Verification Problem as an optimization problem

Now we rewrite the neural network verification problem as a **constrained optimization problem** (still using the simple network example):

$$\min y$$

$$s.t. \quad y = w^{(3)T}\,\hat{z}^{(2)} \qquad \text{(linear layers)}$$

$$\hat{z}^{(2)} = max(z^{(2)}, 0) \qquad \text{(ReLU activation)}$$

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

$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$

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

$$x_i \leq u_i$$

$$x_i \geq li \qquad \text{(element-wise input bounds)}$$

Verification Problem as an optimization problem

Now we rewrite the neural network verification problem as a **constrained optimization problem** (still using the simple network example):

$$\min y$$

$$s.t. \quad y = W^{(3)T}\,\hat{z}^{(2)} \qquad \text{(linear constraints)}$$

$$\hat{z}^{(2)} = max(z^{(2)}, 0) \qquad \text{(nonlinear constraints)}$$

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

$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$

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

$$x_i \leq u_i$$

$$x_i \geq l_i$$

(inputs bounds are also linear constraints)

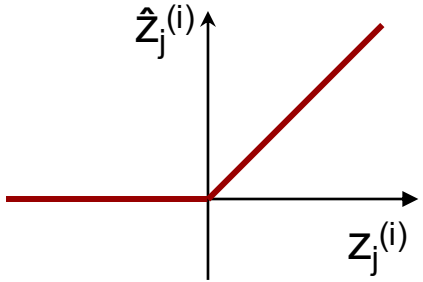
Optimization formulation for ReLU neurons

Let's look at this constraint more carefully: $\hat{z}_j^{(i)} = \max(z_j^{(i)}, 0)$ (for all i, j)

One approach: **mixed integer linear programming (MILP)** formulation

Create an **integer variable** $p_j^{(i)} \in \{0,1\}$ and rewrite using "big" M number:

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \leq M_2 p_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq 0$$



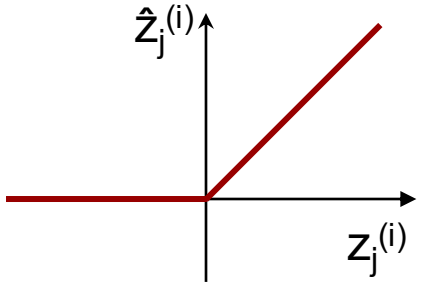
Optimization formulation for ReLU neurons

Let's look at this constraint more carefully: $\hat{z}_j^{(i)} = \max(z_j^{(i)}, 0)$ (for all i, j)

$p_j^{(i)} = 1$ ReLU is active $\hat{z}_j^{(i)} = z_j^{(i)}$; $\hat{z}_j^{(i)} \geq 0$; $\hat{z}_j^{(i)} \leq M_2$; for $M_2 \gg 0$ this is no constraint

$p_j^{(i)} = 0$ ReLU is inactive $\hat{z}_j^{(i)} = 0$; $\hat{z}_j^{(i)} \geq z_j^{(i)}$; $\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1$; for $M_1 \ll 0$ no constraint

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \leq M_2 p_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq 0$$

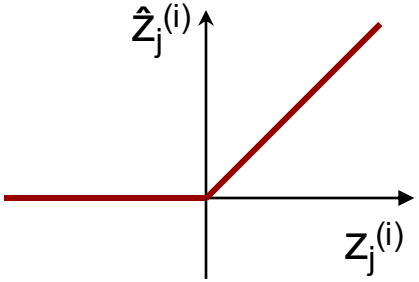


M_1 needs to be small (negative) enough

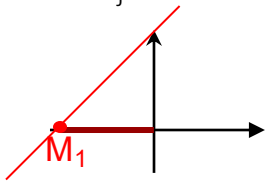
M_2 needs to be large enough

Optimization formulation for ReLU neurons

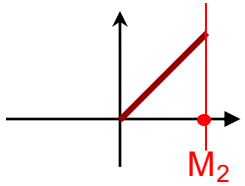
$$\hat{z}_j^{(i)} \le z_j^{(i)} - M_1\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \le M_2 p_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge z_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge 0$$



When $p_j^{(i)} = 0$:



When $p_j^{(i)} = 1$:



$$\hat{z}_j^{(i)} \le z_j^{(i)} - M_1 \quad \hat{z}_j^{(i)} \le 0$$
$$\hat{z}_j^{(i)} \ge z_j^{(i)} \quad \hat{z}_j^{(i)} \ge 0$$
$$\hat{z}_j^{(i)} \le z_j^{(i)} \quad \hat{z}_j^{(i)} \le M_2$$
$$\hat{z}_j^{(i)} \ge z_j^{(i)} \quad \hat{z}_j^{(i)} \ge 0$$

M_1 needs to be small (negative) enough
M_2 needs to be large enough

Pre-activation bounds

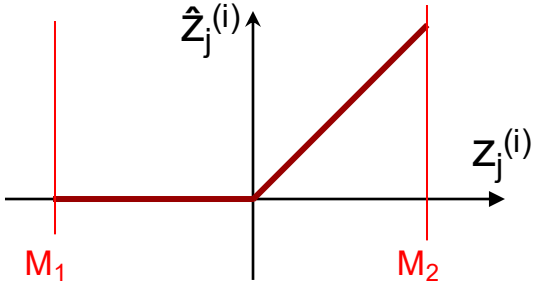
For this to work, M_1 and M_2 must be properly selected.

If set too conservatively, like $M_1 = -100000$ and $M_2 = 100000$, too many constraints will be "active" slowing the solver

How to find appropriate M1 and M2?

We can use optimization to find these pre-activation bounds - the same formulation as our verification problem before, just changing the optimization variables

MILP, LP, or more efficient methods can be used



$$\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \leq M_2 p_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq 0$$

$$l_j^{(i)} = \min z_j^{(i)}$$
$$u_j^{(i)} = max\, z_j^{(i)}$$

Putting it all together with Pre-activation bounds

$$\sout{min\ y}$$

$$s.t. \quad y = w^{(3)T}\hat{z}^{(2)}$$

$$\hat{z}^{(2)} = max(z^{(2)}, 0)$$

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

$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$

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

$$x_i \le u_i$$

$$x_i \ge l_i$$

$$l_j^{(i)} = \min z_j^{(i)}$$

$$u_j^{(i)} = \max z_j^{(i)}$$

We can use optimization to find these pre-activation bounds - the same formulation as our verification problem before, just changing the optimization variables.

Putting it all together with Pre-activation bounds

$$\cancel{min\, y}$$

$$s.t. \quad y = W^{(3)T}\, \hat{z}^{(2)}$$

$$\cancel{\hat{z}^{(2)} = max(z^{(2)}, 0)}$$

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

$$\cancel{\hat{z}^{(1)} = max(z^{(1)}, 0)}$$

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

$$x_i \leq u_i^{(0)}$$

$$x_i \geq l_i^{(0)}$$

Each ReLU is represented as

$$l_j^{(i)} = \min z_j^{(i)}$$

$$u_j^{(i)} = max\, z_j^{(i)}$$

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)}\left(1 - p_j^{(i)}\right)$$

$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq 0$$

$$p_j^{(i)} \in \{0,1\}$$

We can use optimization to find these pre-activation bounds - the same formulation as our verification problem before, just changing the optimization variables.

Putting it all together with Pre-activation bounds

$$\min y$$

$$s.t. \quad y = W^{(3)T}\,\hat{z}^{(2)}$$

$$\hat{z}^{(2)} = max(z^{(2)}, 0)$$

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

$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$

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

$$x_i \le u_i^{(0)}$$

$$x_i \ge l_i^{(0)}$$

Integer variables are still hard!

Each ReLU is represented as

$$l_j^{(i)} = \min z_j^{(i)}$$

$$u_j^{(i)} = max\, z_j^{(i)}$$

$$\hat{z}_j^{(i)} \le z_j^{(i)} - l_j^{(i)}\left(1 - p_j^{(i)}\right)$$

$$\hat{z}_j^{(i)} \le u_j^{(i)} p_j^{(i)}$$

$$\hat{z}_j^{(i)} \ge z_j^{(i)}$$

$$\hat{z}_j^{(i)} \ge 0$$

$$p_j^{(i)} \in \{0,1\}$$

Putting it all together with Pre-activation bounds

$$y^*_{MILP} = \min y$$
$$s.t. \quad y = W^{(3)T}\,\hat{z}^{(2)}$$
$$\hat{z}^{(2)} = max(z^{(2)}, 0)$$
$$z^{(2)} = W^{(2)}\,\hat{z}^{(1)}$$
$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$
$$z^{(1)} = W^{(1)}x$$
$$x_i \le u_i^{(0)}$$
$$x_i \ge l_i^{(0)}$$

Integer variables are still hard!

Each ReLU is represented as

$$l_j^{(i)} = \min z_j^{(i)}$$
$$u_j^{(i)} = \max z_j^{(i)}$$

$$\hat{z}_j^{(i)} \le z_j^{(i)} - l_j^{(i)}\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \le u_j^{(i)} p_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge z_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge 0$$
$$p_j^{(i)} \in \{0,1\}$$

Putting it all together with Pre-activation bounds

$$y^*_{LP} = \min y$$

$$s.t. \quad y = w^{(3)T}\,\hat{z}^{(2)}$$

$$\hat{z}^{(2)} = max(z^{(2)}, 0)$$

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

$$\hat{z}^{(1)} = max(z^{(1)}, 0)$$

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

$$x_i \leq u_i^{(0)}$$

$$x_i \geq l_i^{(0)}$$

Each ReLU is represented as

$$l_j^{(i)} = \min z_j^{(i)}$$

$$u_j^{(i)} = \max z_j^{(i)}$$

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)}\left(1 - p_j^{(i)}\right)$$

$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq 0$$

$$0 \leq p_j^{(i)} \leq 1$$

Linear relaxation of integer constraints

How does this affect the solution?

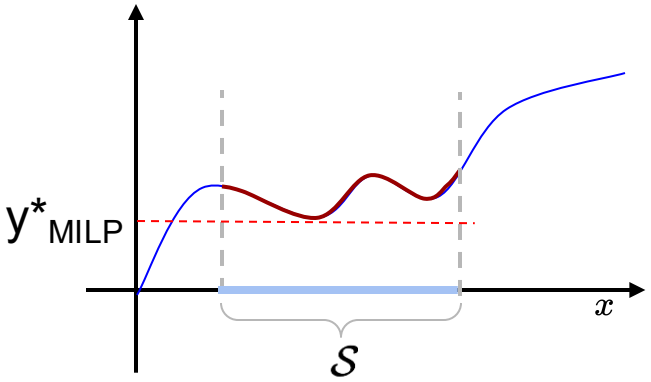
MILP vs LP

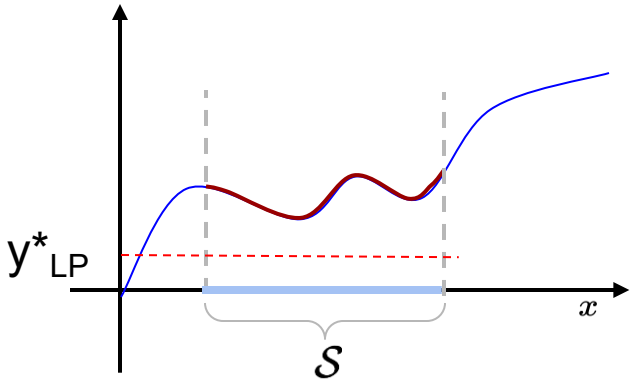
$$y^*_{LP} \leq y^*_{MILP}$$

It's not the original MILP solution, but it is a guaranteed lower bound

Solving LP is (weakly) polynomial time.

Simplex algorithm practically a few orders of magnitude faster



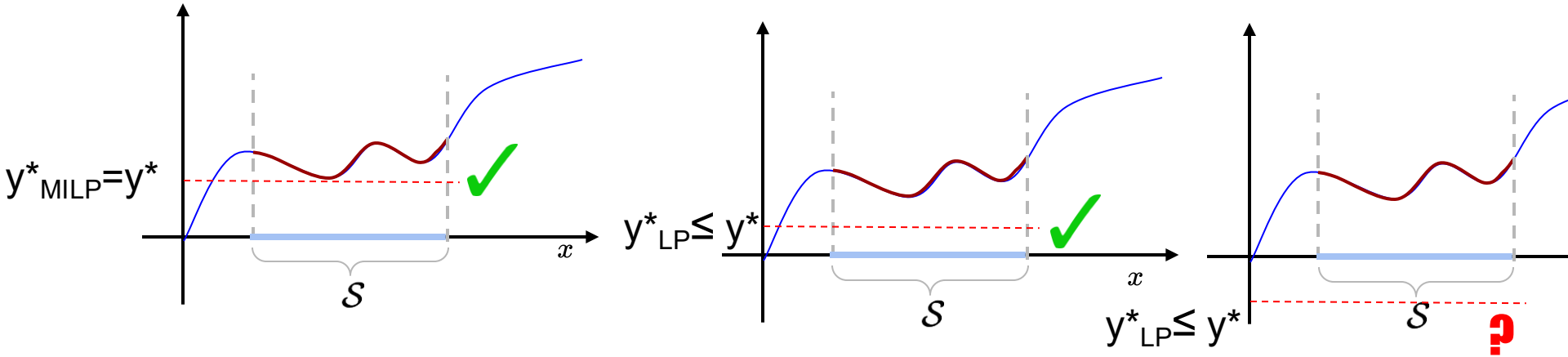


Sound but Incomplete Verification with a Lower Bound

Satisfiability problem: $\exists\, x \in S \;\wedge\; y \leq 0 \;\wedge\; y = f(x)$

If $y^*_{LP} > 0 \Rightarrow y^* > 0 \Rightarrow$ requirement verified (unsatisfiable)

$y^*_{LP} \leq 0 \Rightarrow$ verification result unknown (incomplete)



A closer look at the LP relaxation: "Triangle" relaxation

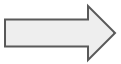
Each ReLU is represented as

$$l_j^{(i)} = \min z_j^{(i)}$$
$$u_j^{(i)} = \max z_j^{(i)}$$

$$\hat{z}_j^{(i)} \le z_j^{(i)} - l_j^{(i)}\left(1 - p_j^{(i)}\right)$$
$$\hat{z}_j^{(i)} \le u_j^{(i)} p_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge z_j^{(i)}$$
$$\hat{z}_j^{(i)} \ge 0$$
$$0 \le p_j^{(i)} \le 1$$

$$p_j^{(i)} \le \frac{\hat{z}_j^{(i)} - z_j^{(i)} + l_j^{(i)}}{l_j^{(i)}}$$

$$p_j^{(i)} \ge \frac{\hat{z}_j^{(i)}}{u_j^{(i)}}$$



Project out p

$$\hat{z}_j^{(i)} \le \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}}$$

(Please note that $l_j^{(i)}$ is negative during the above derivation)

A closer look at the linear programming relaxation

Each ReLU is represented by

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)}(1 - p_j^{(i)})$$

$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

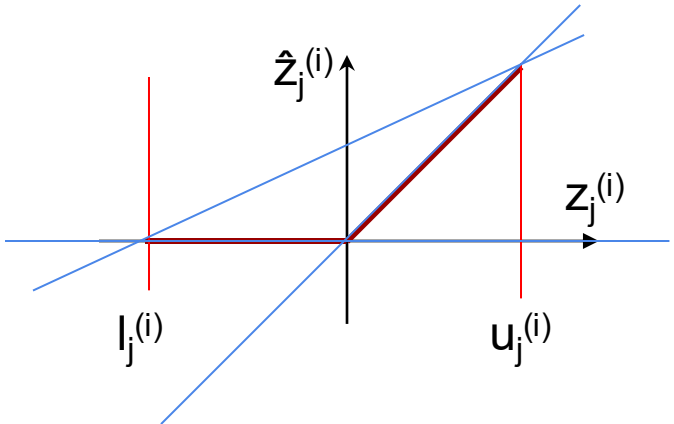
$$\hat{z}_j^{(i)} \geq 0$$

$$\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq 0$$

"Triangle" relaxation



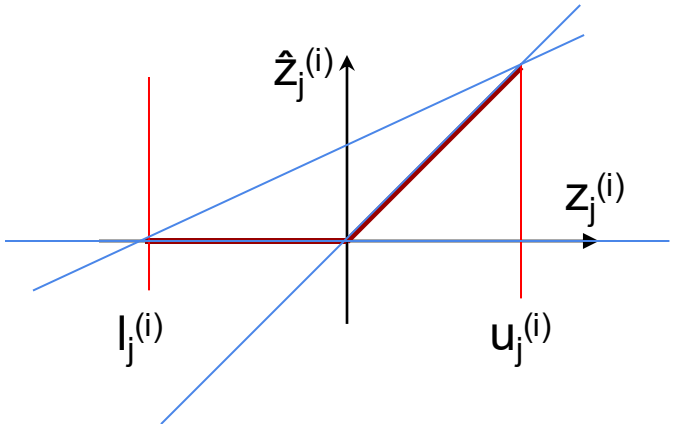
A closer look at the linear programming relaxation

MILP: solutions are constrained on ReLU function
Linear programming: solutions are constrained on the triangle

$$\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

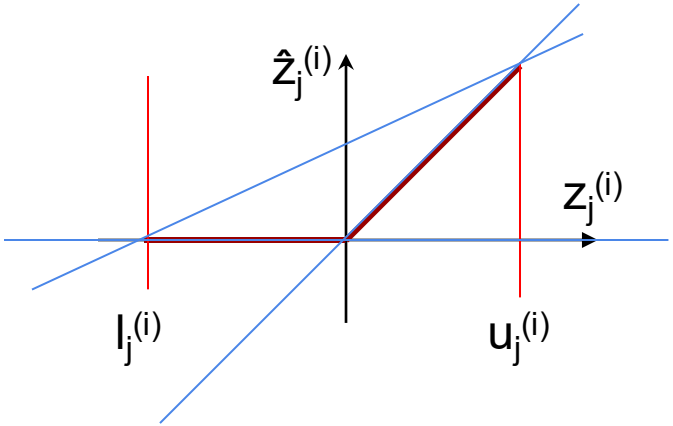
$$\hat{z}_j^{(i)} \geq 0$$

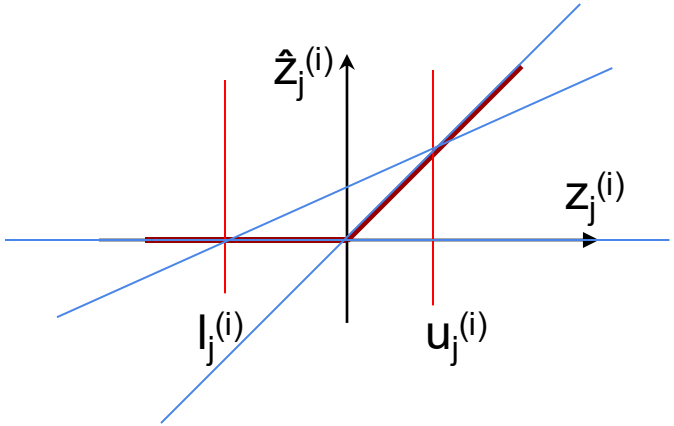


A closer look at the linear programming relaxation

MILP: solutions are constrained on ReLU function
Linear programming: solutions are constrained on the triangle

We want the triangle to be as small as possible! So tight pre-activation bounds are necessary.

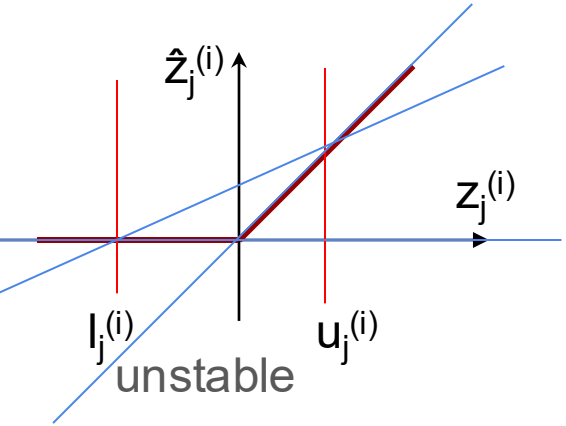


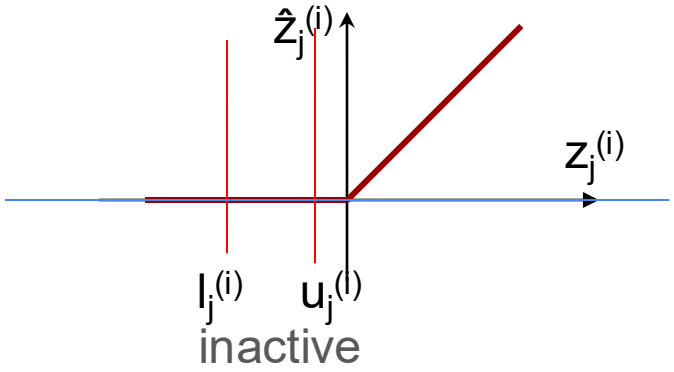


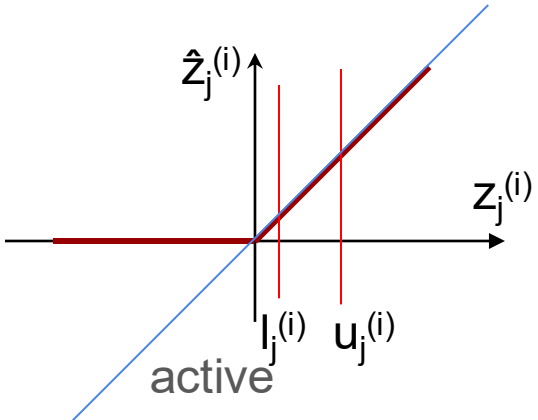
Stable vs. unstable neurons

If bounds are tight enough so $l_j^{(i)} \geq 0$ or $u_j^{(i)} \leq 0$, it is called stable neurons (active or inactive); otherwise it is a unstable neuron

In stable neuron cases, a binary variable in MILP or relaxation in LP is not needed - ReLU becomes a linear function.





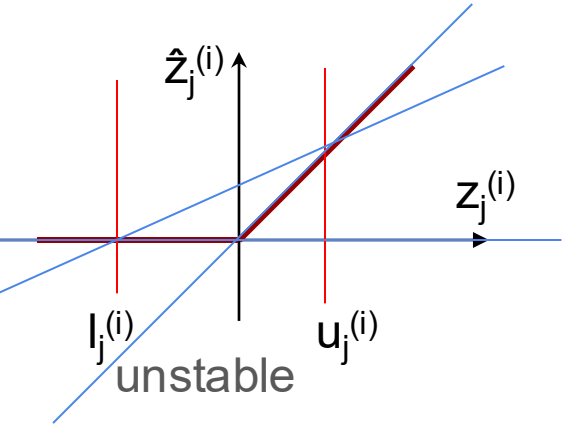


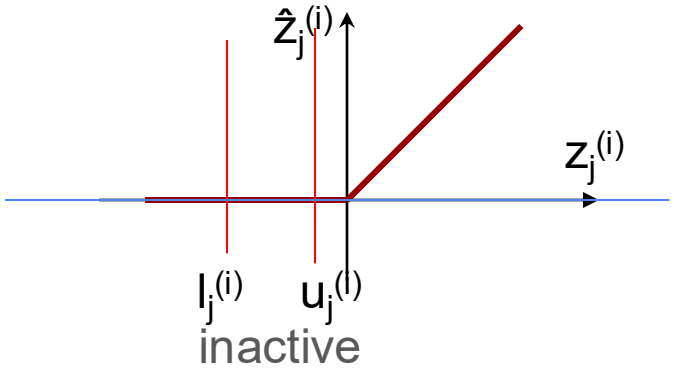
Stable vs. unstable neurons

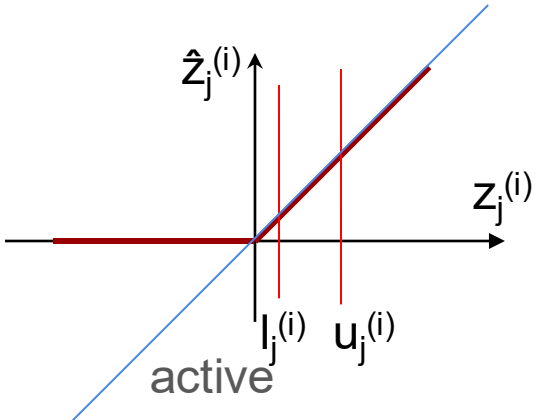
$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)}(1 - p_j^{(i)})$$
$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$
$$\hat{z}_j^{(i)} \geq 0$$

$$\hat{z}_j^{(i)} = 0$$

$$\hat{z}_j^{(i)} = z_j^{(i)}$$







Summary: MILP vs LP vs DPLL(T) in Neural Network Verification

MILP:

- Branch and bound is used to make decisions only on certain number of binary variables
- No decision needed on stable neurons (with the help of pre-activation bounds)
- Specialized methods to accelerate solving (e.g., branching heuristics, cutting planes)
- Complete (solve y* exactly)
- Typically scales much better than DPLL(T)

Linear programming:

- No integer variables
- No variable decisions needed (no exponential time search)
- Simplex algorithm can solve it relatively fast, a few thousands neurons are ok
- Incomplete (has to return unknown in some cases)