

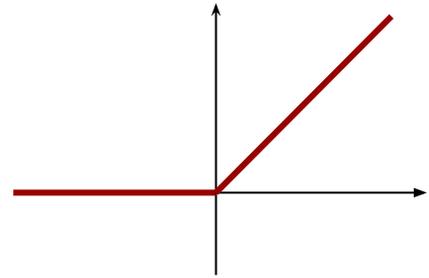
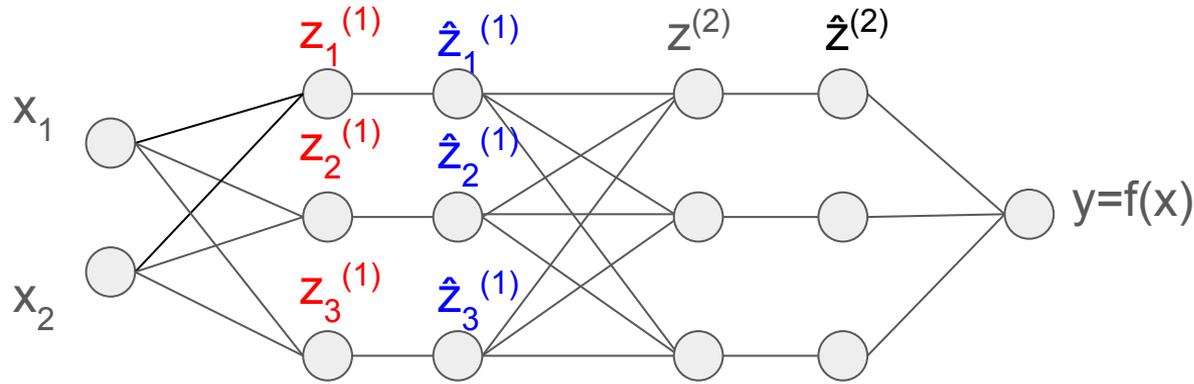
ECE/CS 584: Verification of Embedded and Cyber-physical Systems

Lecture 9: Integer and Linear Programming Formulations for Neural Network Verification

Prof. Huan Zhang

huan@huan-zhang.com

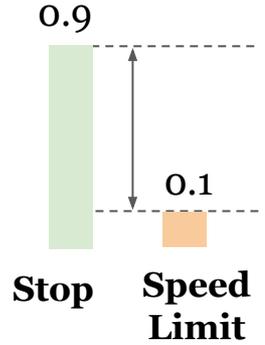
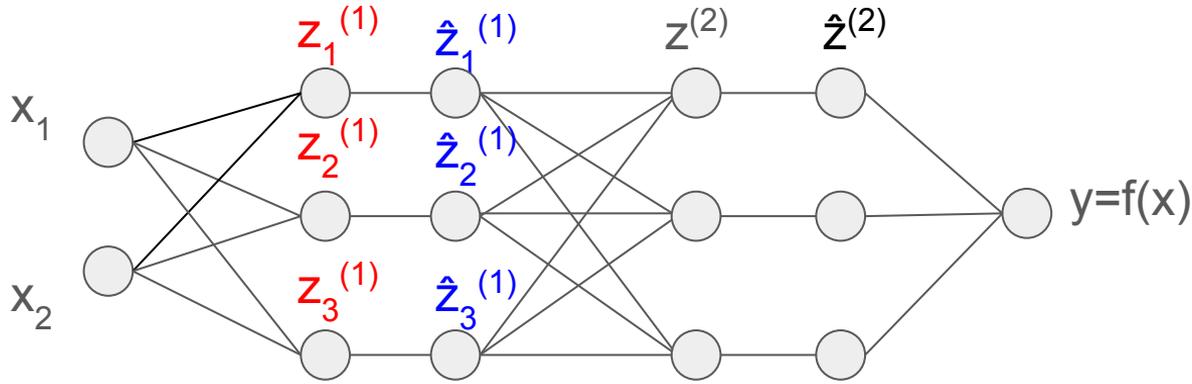
Review: neural networks



Linear layers: $z^{(1)} = W^{(1)} x$ $z^{(2)} = W^{(2)} \hat{z}^{(1)}$ $y = w^{(3)T} \hat{z}^{(2)}$

Nonlinear layers: $\hat{z}_j^{(i)} = \sigma(z_j^{(i)})$ (assume σ is ReLU for now)

Review: neural network verification as a satisfiability problem



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

Input domain under consideration

Negation of the desired property

Defines the neural network

Review: neural network verification as a satisfiability problem

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

$x_i \leq u_i \wedge x_i \geq l_i$ (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

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

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

In general this is very slow! Faster methods in the next a few lectures.

How does a SMT solver solve this problem?

First step: obtain the abstract version of the problem

$$x_i \leq u_i \wedge x_i \geq l_i \quad (\text{assuming box constraints})$$

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

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

How does a SMT solver solve this problem?

Obtain the abstract version of the problem

$$x_i \leq u_i \wedge x_i \geq l_i \quad (\text{assuming box constraints})$$

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

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

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

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

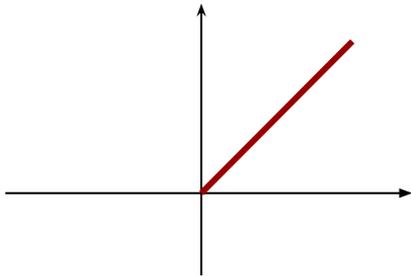
How does a SMT solver solves this problem?

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

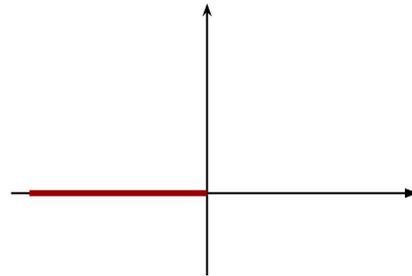
When $p_j^{(i)} = \text{True}$, $q_j^{(i)}$ must be true; $p_j^{(i)} = \text{False}$, $r_j^{(i)}$ must be true;

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

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



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



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

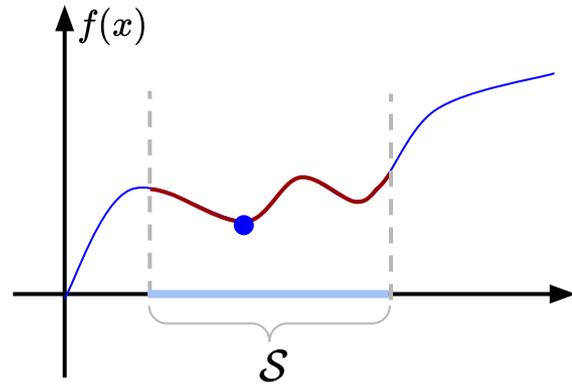
Can we solve the problem without setting every $p_j^{(i)}$?

We will tack this problem from an **optimization** point of view today

Mathematical optimization problems

Given a **objective function** $f: S \rightarrow \mathbb{R}$

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 a **objective function** $f: S \rightarrow \mathbb{R}$

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, ...

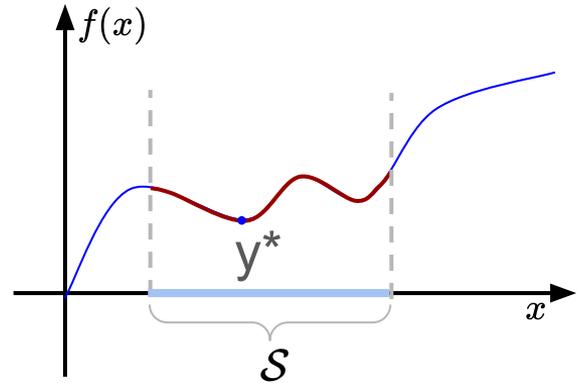
Verification Problem as an optimization problem

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

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

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

If the optimal objective $y^* \leq 0$, 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):

$$\begin{aligned} & \min y \\ \text{s.t. } & y = \mathbf{w}^{(3)\top} \hat{\mathbf{z}}^{(2)} && \text{(linear layers)} \\ & \hat{\mathbf{z}}^{(2)} = \max(\mathbf{z}^{(2)}, 0) && \text{(ReLU activation)} \\ & \mathbf{z}^{(2)} = \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ & \hat{\mathbf{z}}^{(1)} = \max(\mathbf{z}^{(1)}, 0) \\ & \mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} \\ & x_i \leq u_i \\ & x_i \geq l_i && \text{(element-wise input bounds)} \end{aligned}$$

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):

$$\begin{aligned} & \min y \\ \text{s.t. } & y = \mathbf{w}^{(3)\top} \hat{\mathbf{z}}^{(2)} && \text{(linear constraints)} \\ & \hat{\mathbf{z}}^{(2)} = \max(\mathbf{z}^{(2)}, 0) && \text{(nonlinear constraints)} \\ & \mathbf{z}^{(2)} = \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ & \hat{\mathbf{z}}^{(1)} = \max(\mathbf{z}^{(1)}, 0) \\ & \mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} \\ & x_i \leq u_i \\ & x_i \geq l_i && \text{(inputs bounds are also linear constraints)} \end{aligned}$$

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)

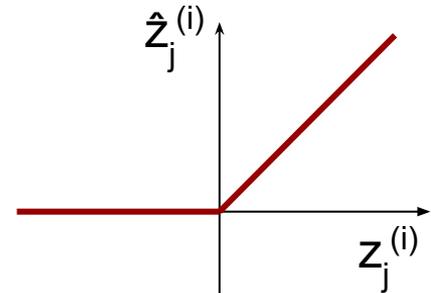
One way to handle it is through the **mixed integer linear programming (MILP)** formulation. Create an **integer variable** $p_j^{(i)} \in \{0, 1\}$, the constraint $\hat{z}_j^{(i)} = \max(z_j^{(i)}, 0)$ can be equivalently written as (here M is a “big” number):

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1(1 - p_j^{(i)})$$

$$\hat{z}_j^{(i)} \leq M_2 p_j^{(i)}$$

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

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



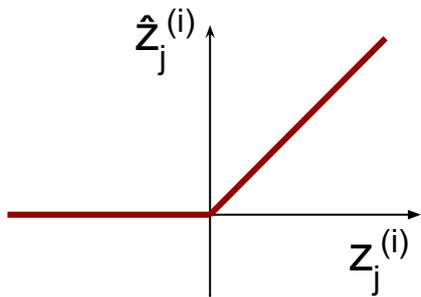
Optimization formulation for ReLU neurons

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - M_1(1 - p_j^{(i)})$$

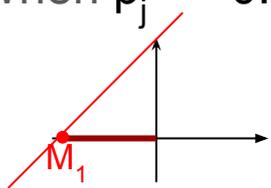
$$\hat{z}_j^{(i)} \leq M_2 p_j^{(i)}$$

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

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



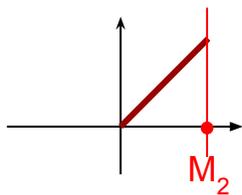
When $p_j^{(i)} = 0$:



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

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

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



$$\hat{z}_j^{(i)} \leq z_j^{(i)} \quad \hat{z}_j^{(i)} \leq M_2$$

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

$$\hat{z}_j^{(i)} \leq M_2$$

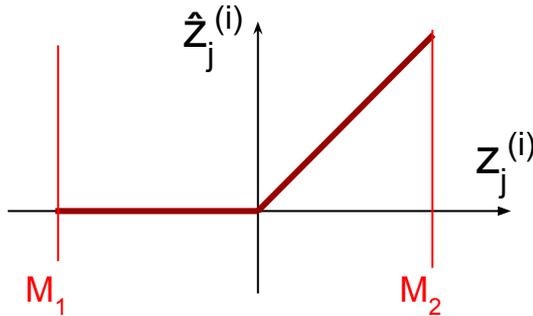
$$\hat{z}_j^{(i)} \geq 0$$

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

Pre-activation bounds

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

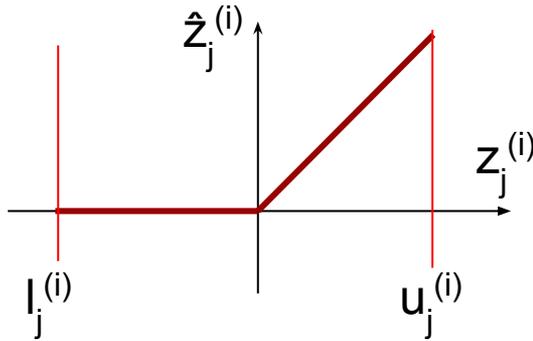
If set too conservatively, like $M_1 = -100000$ and $M_2 = 100000$, solver can be very slow



How to find the tightest M_1 and M_2 ?

Pre-activation bounds

For this formulation to work, M_1 and M_2 must be properly selected (usually denoted as l and u)



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

Let's assume $l_j^{(i)} \leq 0$
and $u_j^{(i)} \geq 0$ for now

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 (next lecture) can be used to find these.

Pre-activation bounds

$$\begin{aligned} & \min_{\mathbf{y}} \\ \text{s.t. } & \mathbf{y} = \mathbf{w}^{(3)\top} \hat{\mathbf{z}}^{(2)} \\ & \hat{\mathbf{z}}^{(2)} = \max(\mathbf{z}^{(2)}, 0) \\ & \mathbf{z}^{(2)} = \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ & \hat{\mathbf{z}}^{(1)} = \max(\mathbf{z}^{(1)}, 0) \\ & \mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} \\ & \mathbf{x}_i \leq u_i^{(0)} \\ & \mathbf{x}_i \geq l_i^{(0)} \end{aligned}$$

$$l_j^{(i)} := \min z_j^{(i)} \quad \text{or} \quad u_j^{(i)} := \max z_j^{(i)}$$

Let's assume $l_j^{(i)} \leq 0$
and $u_j^{(i)} \geq 0$ for now

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

Mixed Integer Linear Programming formulation

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

$$\begin{aligned} \mathbf{y}^*_{\text{MILP}} &:= \min y \\ \text{s.t. } & y = \mathbf{W}^{(3)\top} \hat{\mathbf{z}}^{(2)} \\ & \hat{\mathbf{z}}^{(2)} = \max(\mathbf{z}^{(2)}, \mathbf{0}) \\ & \mathbf{z}^{(2)} = \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ & \hat{\mathbf{z}}^{(1)} = \max(\mathbf{z}^{(1)}, \mathbf{0}) \\ & \mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} \\ & x_i \leq u_i^{(0)} \\ & x_i \geq l_i^{(0)} \end{aligned}$$

Each ReLU is represented by

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

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

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

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

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

Let's assume $l_j^{(i)} \leq 0$
and $u_j^{(i)} \geq 0$ for now

Mixed Integer Linear Programming formulation

However, the integer variables $p_j^{(i)}$ are still hard to handle! (NP-hard)

$$\begin{aligned} \mathbf{y}^*_{\text{MILP}} &:= \min y \\ \text{s.t. } & y = \mathbf{w}^{(3)\top} \hat{\mathbf{z}}^{(2)} \\ & \hat{\mathbf{z}}^{(2)} = \max(\mathbf{z}^{(2)}, \mathbf{0}) \\ & \mathbf{z}^{(2)} = \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ & \hat{\mathbf{z}}^{(1)} = \max(\mathbf{z}^{(1)}, \mathbf{0}) \\ & \mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} \\ & \mathbf{x}_i \leq \mathbf{u}_i^{(0)} \\ & \mathbf{x}_i \geq \mathbf{l}_i^{(0)} \end{aligned}$$

Each ReLU is represented by

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

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

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

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

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

Relaxation of integer variables: Linear programming

Now **relaxing** the integer constraints into continuous ones.

How does this affect the solution?

$$\begin{aligned} \mathbf{y}^*_{\text{LP}} &:= \min y \\ \text{s.t. } y &= \mathbf{w}^{(3)\top} \hat{\mathbf{z}}^{(2)} \\ \hat{\mathbf{z}}^{(2)} &= \max(\mathbf{z}^{(2)}, \mathbf{0}) \\ \mathbf{z}^{(2)} &= \mathbf{W}^{(2)} \hat{\mathbf{z}}^{(1)} \\ \hat{\mathbf{z}}^{(1)} &= \max(\mathbf{z}^{(1)}, \mathbf{0}) \\ \mathbf{z}^{(1)} &= \mathbf{W}^{(1)} \mathbf{x} \\ x_i &\leq u_i^{(0)} \\ x_i &\geq l_i^{(0)} \end{aligned}$$

Each ReLU is represented by

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

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

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

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

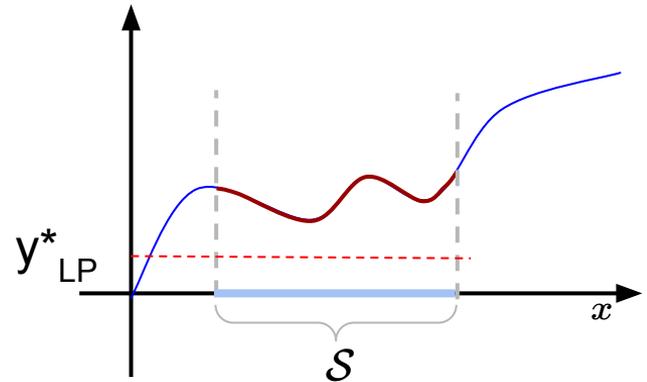
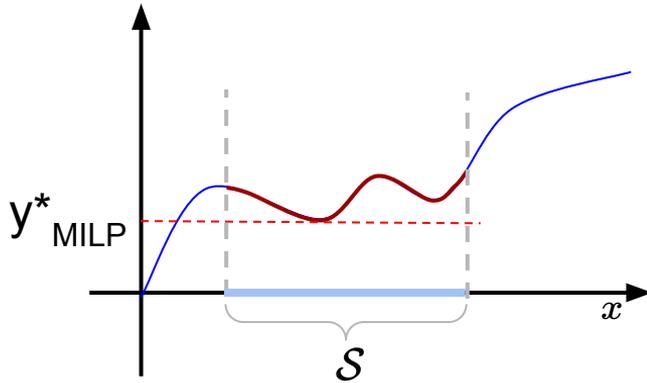
$$p_j^{(i)} \in \{0, 1\} \quad 0 \leq p_j^{(i)} \leq 1$$

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 polynomial time. Practically a few orders of magnitude faster.

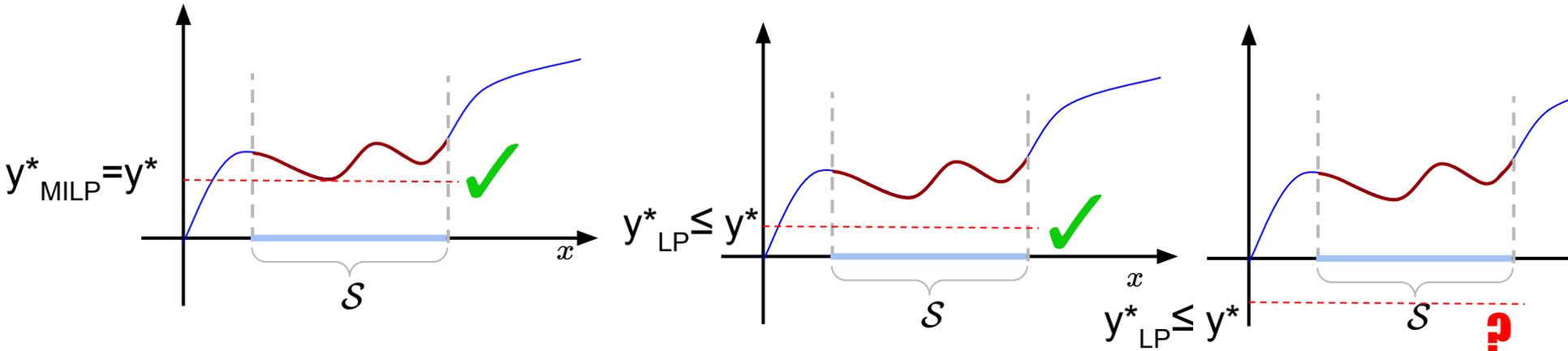


Sound but Incomplete Verification with a Lower Bound

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

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

$y_{LP}^* \leq 0 \Rightarrow$ return unknown (incomplete)



A closer look at the linear programming relaxation

Each ReLU is represented by

$$\begin{aligned} \hat{z}_j^{(i)} &\leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)}) \\ \hat{z}_j^{(i)} &\leq u_j^{(i)} p_j^{(i)} \\ \hat{z}_j^{(i)} &\geq z_j^{(i)} \\ \hat{z}_j^{(i)} &\geq 0 \\ 0 &\leq p_j^{(i)} \leq 1 \end{aligned} \quad \longrightarrow \quad \begin{aligned} p_j^{(i)} &\leq \frac{\hat{z}_j^{(i)} - z_j^{(i)} + l_j^{(i)}}{l_j^{(i)}} \\ p_j^{(i)} &\geq \frac{\hat{z}_j^{(i)}}{u_j^{(i)}} \end{aligned} \quad \longrightarrow \quad \begin{aligned} \hat{z}_j^{(i)} &\leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} \end{aligned}$$

Project out p

(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

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

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

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

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

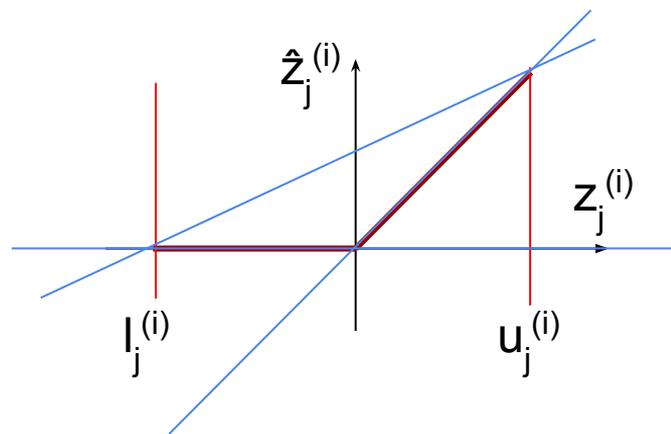


$$\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}}$$

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

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

“Triangle” relaxation



A closer look at the linear programming relaxation

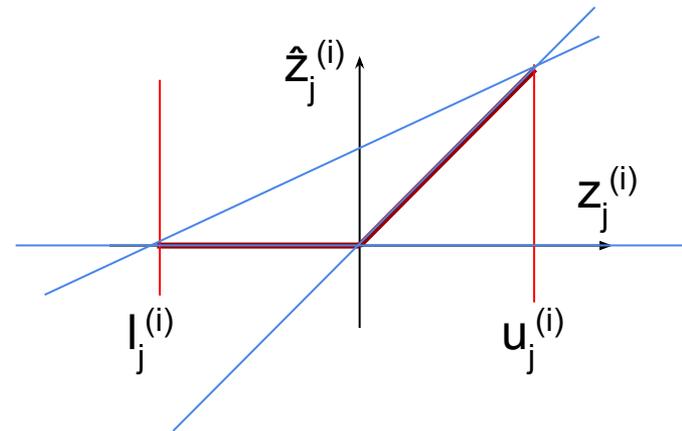
MILP: solutions are constrained on **ReLU function**

Linear programming: solutions are constrained on the **triangle**

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

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

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

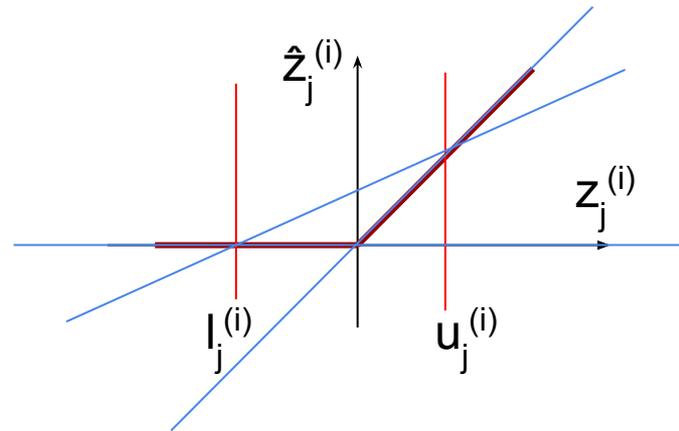
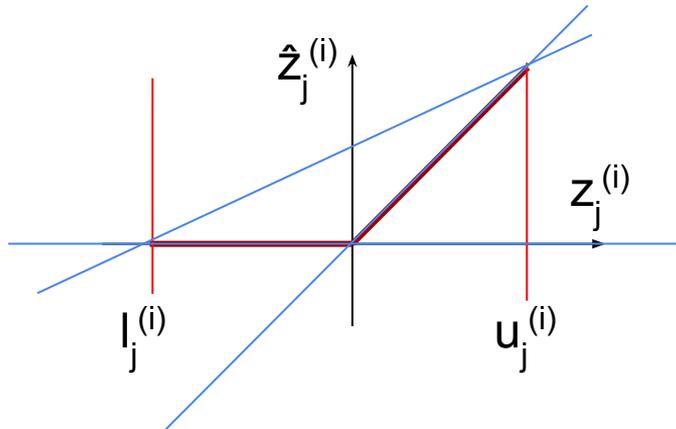


A closer look at the linear programming relaxation

MILP: solutions are constrained on **ReLU function**

Linear programming: solutions are constrained on the **triangle**

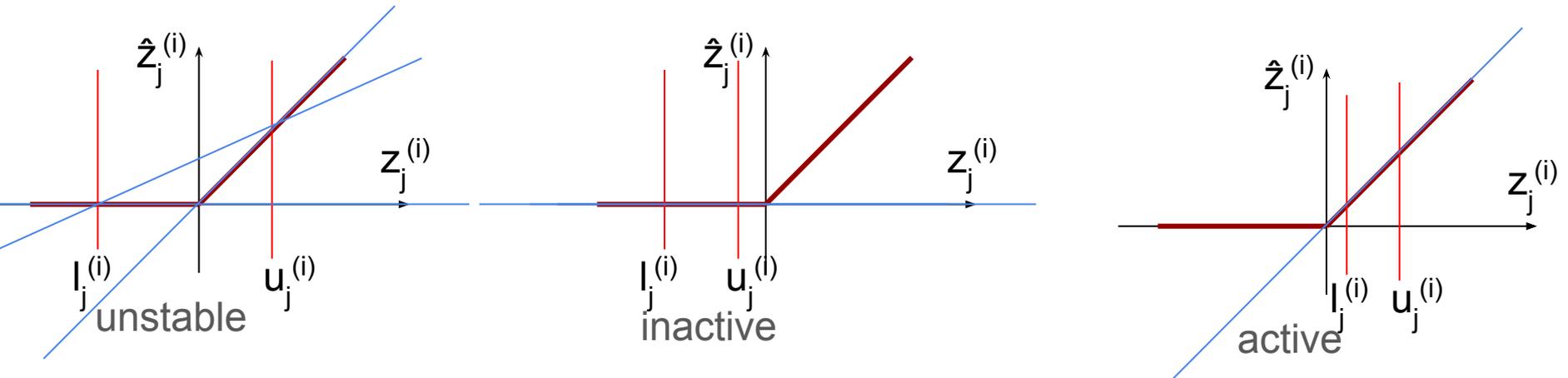
We want the triangle to be as small as possible! So tight pre-activation bounds are necessary.



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

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

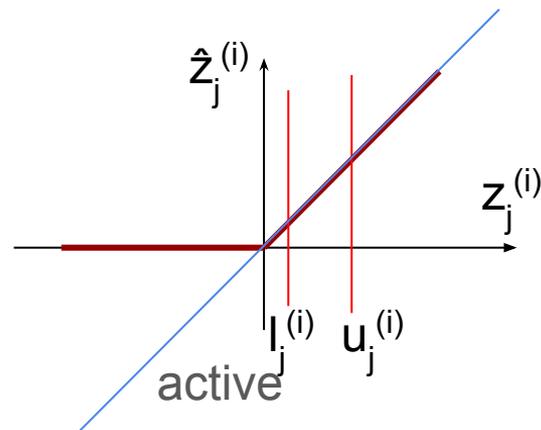
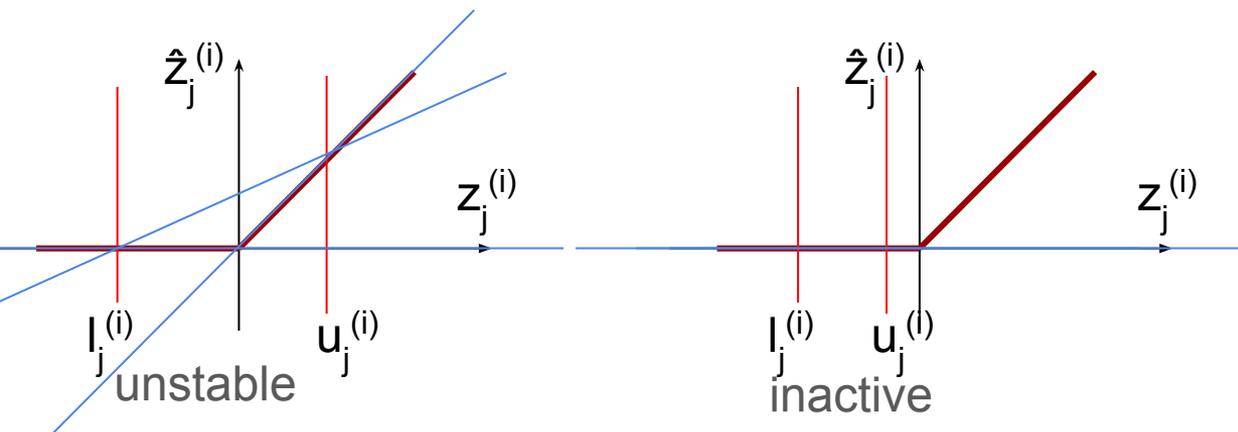
$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$

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

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

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

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



MILP vs LP vs DPLL(T)

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)

Summary

- Neural network verification problem
- Solving the verification problem with SMT solvers
- Integer programming formulation
- Linear programming formulation and linear relaxation of ReLUs
- Next lecture: linear bound propagation method (CROWN) for efficient neural network verification
- **Reading for the next lecture:**
 - <http://arxiv.org/pdf/1811.00866.pdf>
 - <https://arxiv.org/pdf/1902.08722.pdf>