Reliable and Trustworthy Artificial Intelligence

Lecture 3: MILP and DeepPoly for Certification

Martin Vechev
ETH Zurich

Fall 2022

SRILAB

http://www.sri.inf.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture Outline

- Complete verification with classic MILP (Mixed Integer Linear Program), incorporating Box constraints.

- Advanced convex relaxation: DeepPoly.

Lecture Outline

- **Complete verification with classic MILP (Mixed Integer Linear Program), incorporating Box constraints.**

- Advanced convex relaxation: DeepPoly.

Complete method: MILP

Let us incorporate MILP (Mixed Integer Linear Program) solvers in order to verify robustness. For ReLU networks, MILP is complete. However, this comes at the cost that MILP is at least NP-complete (cannot be solved in polynomial time in any known manner).

We will first define the standard MILP problem. Then, we will use the Box bounds to constrain MILP. So, Box will help MILP be faster by doing less work (MILP still remains complete).

General MILP

$$\boldsymbol{min\ c_1x_1 + c_2x_2 + \cdots + c_nx_n}$$

objective

$$\boldsymbol{a_{11}x_1 + \cdots + a_{1n}x_n \le b_1}$$

....

$$\boldsymbol{a_{m1}x_1 + \cdots + a_{mn}x_n \le b_m}$$

constraints

$$\boldsymbol{x_j} \in \boldsymbol{Z} \qquad \boldsymbol{x_i} \in \boldsymbol{R}$$

some $\boldsymbol{x_j}$ are integer

some $\boldsymbol{x_i}$ are real

- $\boldsymbol{c_i, a_{ij}, b_i} \in \boldsymbol{R}$
- some $\boldsymbol{x_j}$ ‘s can be integers (or even binary), hence Mixed-Integer problem

MILP + Box Bounds

$$\min c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

objective

$$a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1$$
....
$$a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m$$

constraints

$$l_i \leq x_i \leq u_i \quad 1 \leq i \leq n$$

bounds on continuous x_i

$$x_j \in Z \qquad x_i \in R$$

some x_j are integer
some x_i are real

- $c_i, a_{ij}, b_i \in R$ **and** $l, u \in R^n$
- some x_j ‘s can be integers (or even binary), hence Mixed-Integer problem
- state-of-the-art solvers (e.g., Gurobi) benefit from bounds on x_i ’s

Encoding introduced in: Evaluating Robustness of Neural Networks with Mixed Integer Programming, ICLR 2019 (see Appendix)
https://openreview.net/pdf?id=HyGIdiRqtm

MILP encoding of Neural Network

To encode the network as a MILP instance, we need to:

1. Encode the Affine layer
2. Encode the ReLU layer
3. Encode the pre-condition $\boldsymbol{\phi}$
4. Encode the post-condition $\boldsymbol{\psi}$

Encode Affine layer as MILP

This is direct as it is just a linear constraint

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

where $\boldsymbol{W}$ are the weights and $\boldsymbol{b}$ is the bias

(note: convolution is also an affine transformation, can be encoded directly)

Encode ReLU layer as MILP

ReLU definition is:

$$\boldsymbol{y} = \boldsymbol{ReLU}(\boldsymbol{x}) = \boldsymbol{max}(\mathbf{0}, \boldsymbol{x})$$

MILP ReLU encoding is:

$$\boldsymbol{y} \le \boldsymbol{x} - \boldsymbol{l} * (\mathbf{1} - \boldsymbol{a})$$

$$\boldsymbol{y} \ge \boldsymbol{x}$$

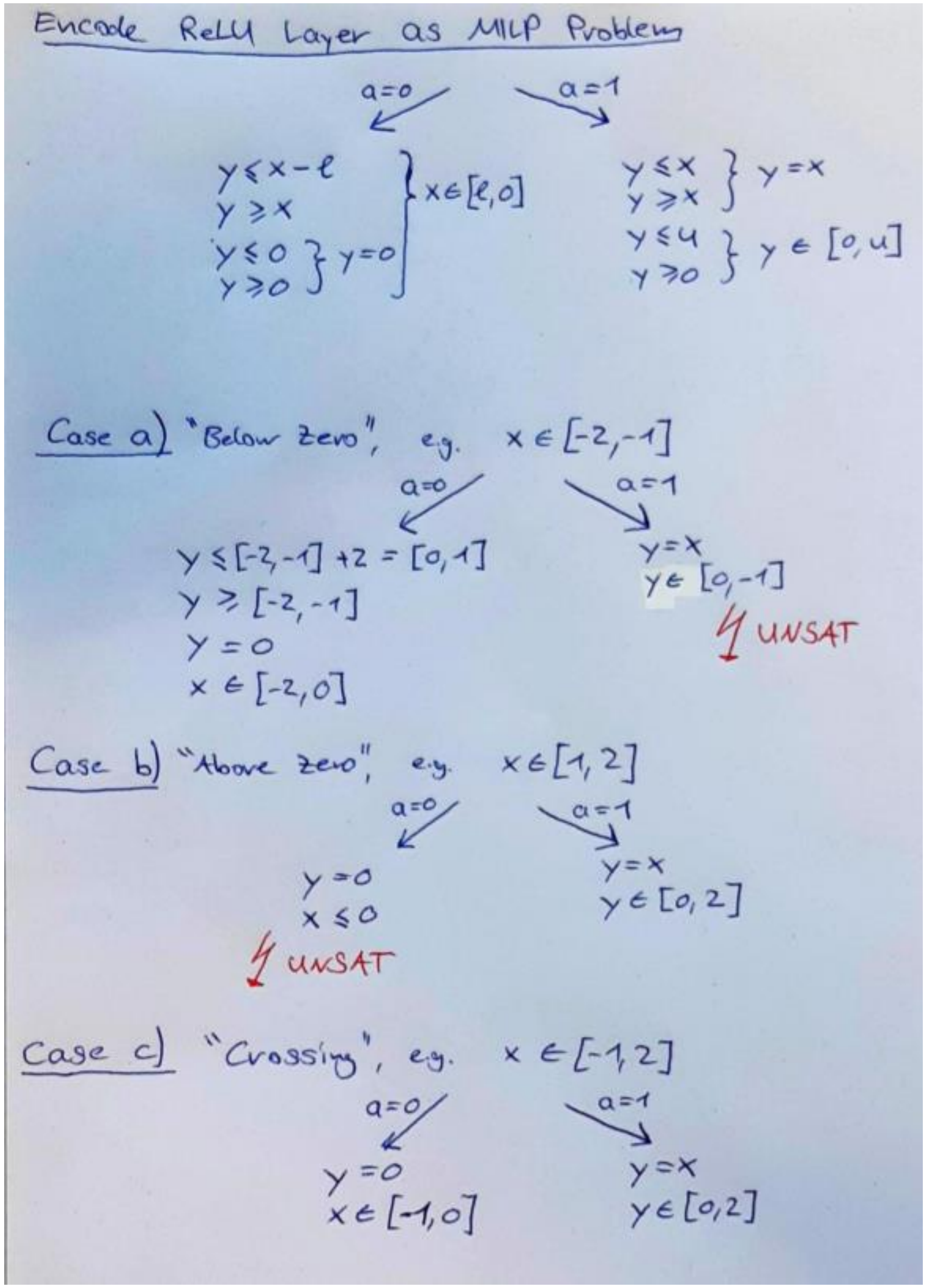
$$\boldsymbol{y} \le \boldsymbol{u} * \boldsymbol{a}$$

$$\boldsymbol{y} \ge \mathbf{0}$$

$\boldsymbol{a} \in \{\mathbf{0}, \mathbf{1}\}$ $\longrightarrow$ $\boldsymbol{a}$ is a binary integer variable

This assumes we have computed lower $\boldsymbol{l}$ and upper $\boldsymbol{u}$ bounds for the input neuron $\boldsymbol{x}$ (e.g., by using Box beforehand).

(*Intuition on next slide)



Encode ReLU Layer as MILP Problem

$$\begin{tikzcd} & \cdot \arrow[dl, "a=0"'] \arrow[dr, "a=1"] & \\ \text{(a=0)} & & \text{(a=1)} \end{tikzcd}$$

$a=0$:

$y \leq x - l$, $y \geq x$ $\}$ $x \in [l, 0]$

$y \leq 0$, $y \geq 0$ $\}$ $y = 0$

$a=1$:

$y \leq x$, $y \geq x$ $\}$ $y = x$

$y \leq u$, $y \geq 0$ $\}$ $y \in [0, u]$

Case a) "Below zero", e.g. $x \in [-2,-1]$

$a=0$:

$y \leq [-2,-1] + 2 = [0,1]$

$y \geq [-2,-1]$

$y = 0$

$x \in [-2, 0]$

$a=1$:

$y = x$

$y \in [0,-1]$

↯ UNSAT

Case b) "Above zero", e.g. $x \in [1, 2]$

$a=0$:

$y = 0$

$x \leq 0$

↯ UNSAT

$a=1$:

$y = x$

$y \in [0, 2]$

Case c) "Crossing", e.g. $x \in [-1, 2]$

$a=0$:

$y = 0$

$x \in [-1, 0]$

$a=1$:

$y = x$

$y \in [0,2]$

Reminder: ReLU encoding

$$\boldsymbol{y \leq x - l * (1 - a)}$$

$$\boldsymbol{y \geq x}$$

$$\boldsymbol{y \leq u * a}$$

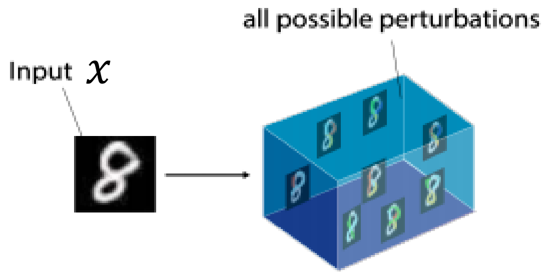
$$\boldsymbol{y \geq 0}$$

$$\boldsymbol{a \in \{0, 1\}}$$

Encode Pre-condition $\boldsymbol{\phi}$ as MILP

Lets take $\boldsymbol{\phi}$ = L_∞ ball around x as an example:

L_∞ ball: Ball$(x)_\epsilon$ = $\{x' \mid \; || x - x' ||_\infty < \epsilon \}$



MILP encoding:

$$x_i - \epsilon \leq x' \leq x_i + \epsilon \longrightarrow$$

That is, we will introduce lower and upper bound constraints for each input neuron x_i'

Encode Post-condition $\boldsymbol{\psi}$ as MILP

Lets take $\boldsymbol{\psi}$ = label 0 is more likely than label 1 (our example network)

$$\psi = o_0 > o_1 \longrightarrow$$

We want to prove this. Hence, this must be forming our MILP objective

MILP encoding:

$$\boldsymbol{min}\ o_0 - o_1$$

After MILP finishes, final verification takes the computed values for o_0 , o_1 and checks if o_0 is indeed greater than o_1 . If yes, verification succeeds, if not, verification fails.

Generic vs. Instantiated MILP

Generic MILP problem

$$\boldsymbol{min\ c_1 x_1 + c_2 x_2 + \cdots + c_n x_n}$$

$$\boldsymbol{a_{11} x_1 + \cdots + a_{1n} x_n \le b_1}$$
....
$$\boldsymbol{a_{m1} x_1 + \cdots + a_{mn} x_n \le b_m}$$

$$\boldsymbol{l_i \le x_i \le u_i \quad 1 \le i \le n}$$

$$\boldsymbol{x_j \in Z}$$

Our MILP instance

$$\boldsymbol{min}\ o_0 - o_1$$

Plug in the Affine and ReLU MILP encodings as defined

$$l_i \le x_i^p \le u_i$$

pre-computed Box bounds on all neurons in each layer p

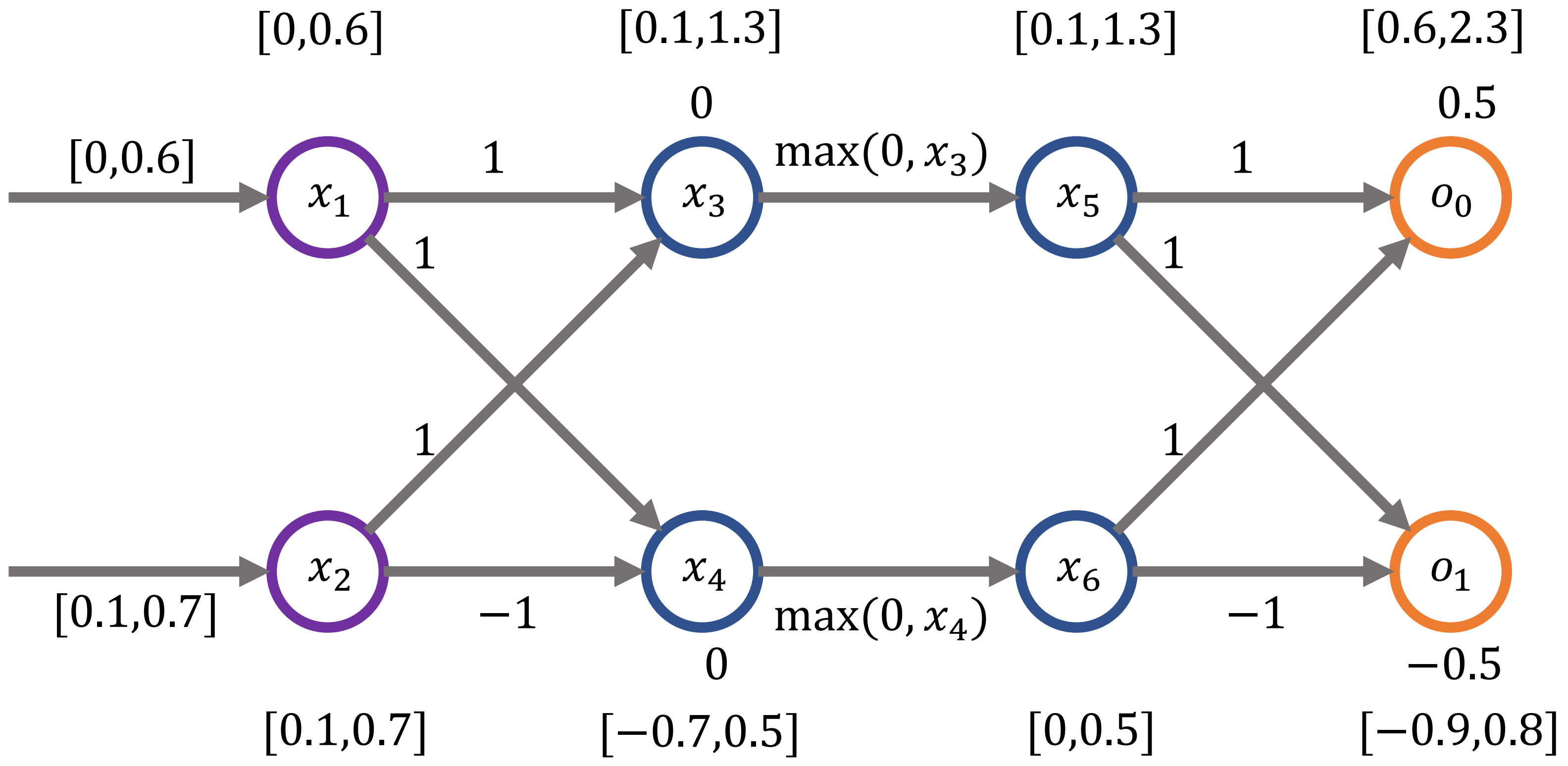
$$x_i - \epsilon \le x_i' \le x_i + \epsilon$$

ϕ: bounds on input neurons

$$\boldsymbol{a_j} \in \{\boldsymbol{0}, \boldsymbol{1}\}$$

These are the $\boldsymbol{a} \in \{\boldsymbol{0}, \boldsymbol{1}\}$ vars from the ReLU encoding

Reminder: these are the bounds with Box



MILP Instance for this network

$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $x_5 = \max(0, x_3)$

$$x_5 \leq x_3 - 0.1 * (1 - a_5)$$
$$x_5 \geq x_3$$
$$x_5 \leq 1.3 * a_5$$
$$x_5 \geq 0$$

ReLU: $x_6 = \max(0, x_4)$

$$x_6 \leq x_4 + 0.7 * (1 - a_6)$$
$$x_6 \geq x_4$$
$$x_6 \leq 0.5 * a_6$$
$$x_6 \geq 0$$

Input bounds

$$0 \leq x_1 \leq 0.6$$
$$0.1 \leq x_2 \leq 0.7$$

Pre-computed Box bounds

$$0.1 \leq x_3 \leq 1.3$$
$$-0.7 \leq x_4 \leq 0.5$$
$$0.1 \leq x_5 \leq 1.3$$
$$0 \leq x_6 \leq 0.5$$

Binary integer variables

$$\boldsymbol{a_5, a_6} \in \{\boldsymbol{0, 1}\}$$

Solving this MILP instance will lead to proving the property

Lets solve MILP to see why Box helps

(Note: MILP solvers employ elaborate algorithms, the idea here is to show how Box can help even state-of-the-art MILP solvers)

Case $\mathbf{a_5 = 0}$

$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $\mathrm{x}_5 = \max(0, x_3)$

$$x_5 \le x_3 - 0.1$$
$$x_5 \ge x_3$$
…

ReLU: $\mathrm{x}_6 = \max(0, x_4)$

$$x_6 \le x_4 + 0.7 * (1 - a_6)$$
$$x_6 \ge x_4$$
$$x_6 \le 0.5 * a_6$$
$$x_6 \ge 0$$

Pre-computed Box bounds

$$0.1 \le x_3 \le 1.3$$
$$-0.7 \le x_4 \le 0.5$$
$$0.1 \le x_5 \le 1.3$$
$$0 \le x_6 \le 0.5$$

This is an infeasible LP instance so MILP does not have to consider it .

Here, Box bounds helped MILP in not exploring further values for $\boldsymbol{a_6}$. So it saved generating two cases for$\boldsymbol{a_6}$.

In practice, we directly generate this MILP

$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $x_5 = \max(0, x_3)$

$$x_5 = x_3$$
$$0.1 \leq x_5 \leq 1.3$$

ReLU: $x_6 = \max(0, x_4)$

$$x_6 \leq x_4 + 0.7 * (1 - a_6)$$
$$x_6 \geq x_4$$
$$x_6 \leq 0.5 * a_6$$
$$x_6 \geq 0$$

can prove x_3 is positive using Box bounds

Input bounds

$$0 \leq x_1 \leq 0.6$$
$$0.1 \leq x_2 \leq 0.7$$

Pre-computed Box bounds

$$0.1 \leq x_3 \leq 1.3$$
$$-0.7 \leq x_4 \leq 0.5$$
$$0.1 \leq x_5 \leq 1.3$$
$$0 \leq x_6 \leq 0.5$$

Binary integer variable

$$\boldsymbol{a_6} \in \{\boldsymbol{0}, \boldsymbol{1}\}$$

Case $\boldsymbol{a_6 = 0}$

$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $\mathrm{x}_5 = \max(0, x_3)$

$$x_5 = x_3$$
$$0.1 \leq x_5 \leq 1.3$$

ReLU: $\mathrm{x}_6 = \max(0, x_4)$

$$x_4 \leq x_6 \leq x_4 + 0.7$$
$$x_6 = 0$$

simplifying

$$o_0 - o_1 = 2 * x_6 + 1$$
$$o_0 - o_1 = 1$$

Input bounds

$$0 \leq x_1 \leq 0.6$$
$$0.1 \leq x_2 \leq 0.7$$

Pre-computed Box bounds

$$0.1 \leq x_3 \leq 1.3$$
$$-0.7 \leq x_4 \leq 0.5$$
$$0.1 \leq x_5 \leq 1.3$$
$$0 \leq x_6 \leq 0.5$$

Here, we proved, subject to the constraints, that the minimum is always 1, hence property holds.

Note: cannot just use the bounds for x_5 and x_6 for computing $o_0 - o_1$: this will be too imprecise! Good to try.

Case $\boldsymbol{a_6 = 1}$

$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $\mathrm{x}_5 = \max(0, x_3)$

$$x_5 = x_3$$
$$0.1 \leq x_5 \leq 1.3$$

ReLU: $\mathrm{x}_6 = \max(0, x_4)$

$$x_6 = x_4$$
$$0 \leq x_6 \leq 0.5$$

simplifying

$$o_0 - o_1 = 2 * x_6 + 1$$
$$1 \leq o_0 - o_1 \leq 2$$
$$min(o_0 - o_1) = 1$$

Input bounds

$$0 \leq x_1 \leq 0.6$$
$$0.1 \leq x_2 \leq 0.7$$

Pre-computed Box bounds

$$0.1 \leq x_3 \leq 1.3$$
$$-0.7 \leq x_4 \leq 0.5$$
$$0.1 \leq x_5 \leq 1.3$$
$$0 \leq x_6 \leq 0.5$$

Here, we proved the property again

Lecture Outline

- Complete verification with classic MILP (Mixed Integer Linear Program), incorporating Box constraints.

- **Advanced convex relaxation: DeepPoly.**

The problem with Box

Box is simple and efficient, but the problem is that it looses too much precision, in both, the ReLU abstract transformer but also in the affine abstract transformer.

The Polyhedral convex relaxation we study now is such that the affine transformer will not lose precision (that is, it will be precise); however, its ReLU transformer will again lose some precision.

DeepPoly convex relaxation: the shape

[Singh et. al, POPL'19]

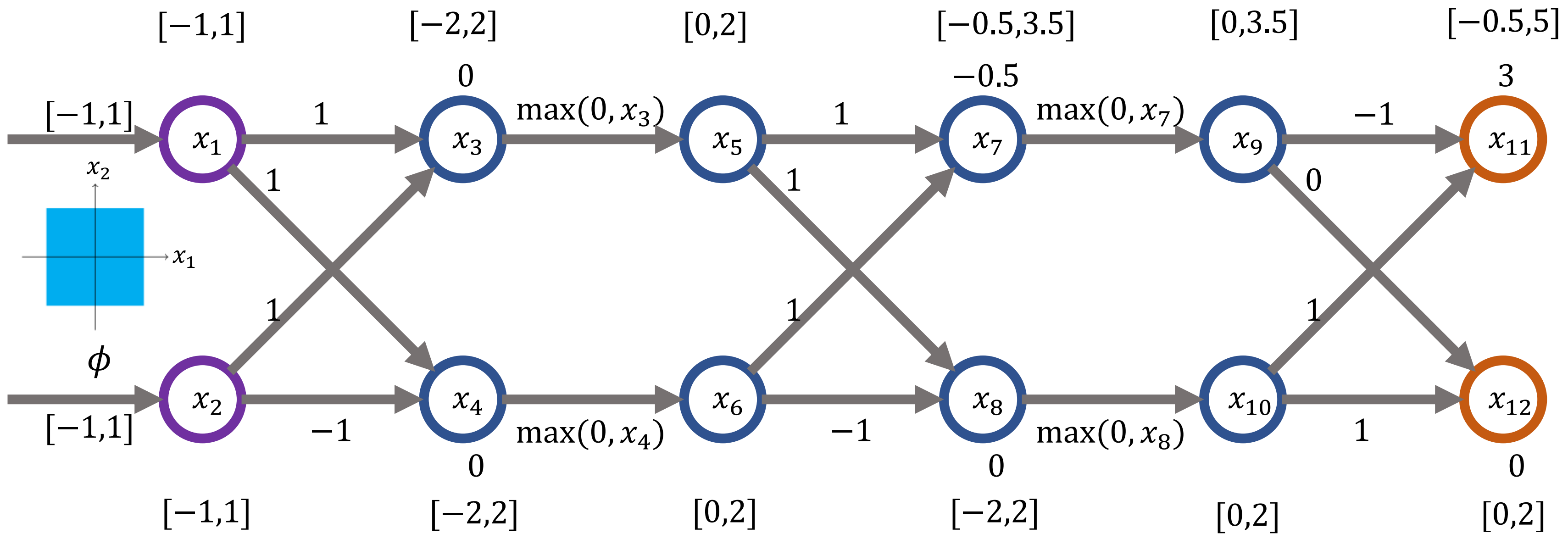
Shape:

for each x_i, we keep:

- An interval constraint: lower bound $l_i \leq x_i$ and upper bound $x_i \leq u_i$
- Two relational constraints: $a_i^{\leq} \leq x_i$ and $x_i \leq a_i^{\geq}$
  where the expressions $a_i^{\leq}, a_i^{\geq}$ are of the form $\sum_j w_j \cdot x_j + v$

- less precise than Polyhedra, restriction needed to ensure scalability
- captures affine transformations precisely
- custom transformers for ReLU, sigmoid, tanh, and maxpool activations

Box relaxation (scalable but imprecise)



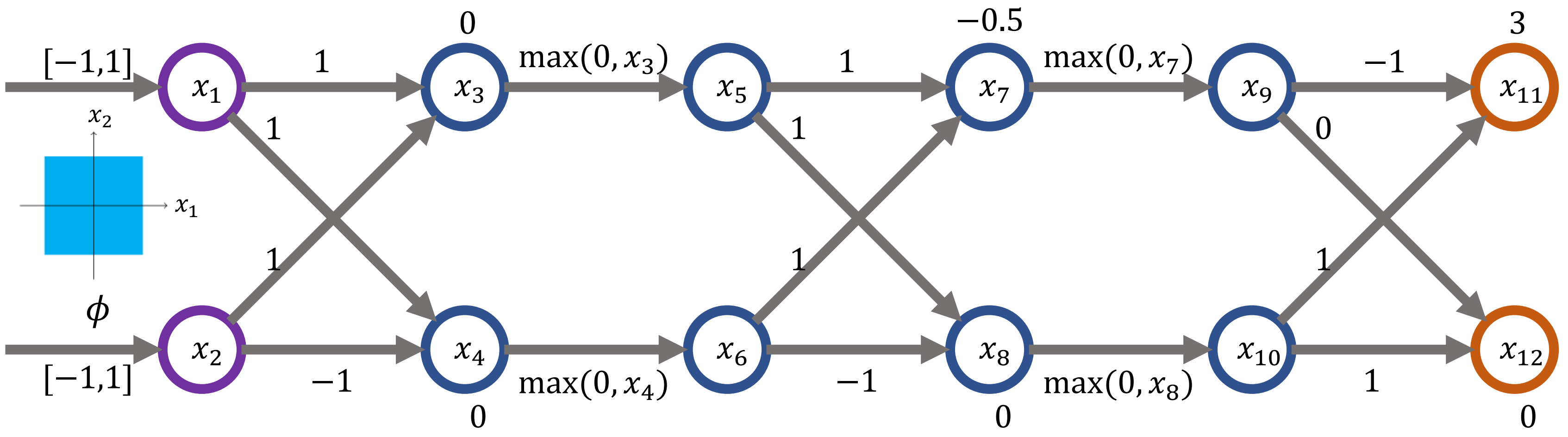
ψ: we want to prove that $x_{11} > x_{12}$ for all values of x_1, x_2 in the input set

Certification with Box fails as it cannot capture relational information

DeepPoly relaxation

$x_1 \geq -1,$
$x_1 \leq 1,$
$l_1 = -1,$
$u_1 = 1$

$x_3 \geq x_1 + x_2,$
$x_3 \leq x_1 + x_2,$
$l_3 = -2,$
$u_3 = 2$



$x_2 \geq -1,$
$x_2 \leq 1,$
$l_2 = -1,$
$u_2 = 1$

$x_4 \geq x_1 - x_2,$
$x_4 \leq x_1 - x_2,$
$l_4 = -2,$
$u_4 = 2$

ReLU activation: $x_j \coloneqq \max(0, x_i)$

Single-neuron transformer for $x_j \coloneqq max(0, x_i)$ that uses l_i, u_i

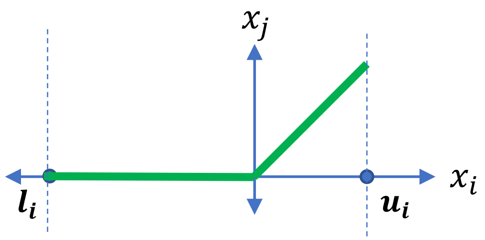
ReLU activation: $x_j \coloneqq \max(0, x_i)$

Single-neuron transformer for $x_j \coloneqq max(0, x_i)$ that uses l_i, u_i

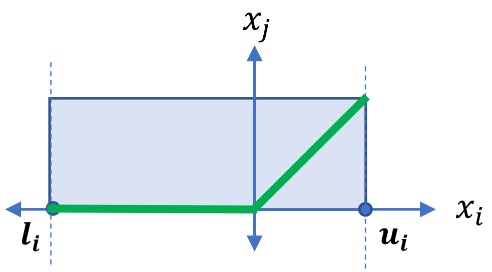
- $if\ u_i \leq 0$: $a_j^{\leq} = a_j^{\geq} = 0, l_j = u_j = 0$ (strictly negative)
- $if\ l_i \geq 0$: $a_j^{\leq} = a_j^{\geq} = x_i, l_j = l_i, u_j = u_i$ (strictly positive)
- $if\ l_i < 0\ and\ u_i > 0$ (crossing ReLU)

Lets discuss the crossing ReLU activation next

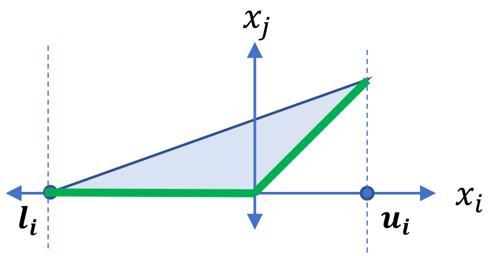
ReLU activation: $x_j := \max(0, x_i)$



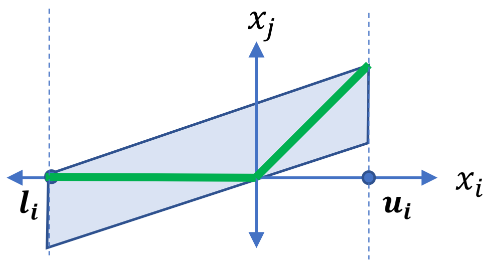
Exact [Katz et al., CAV’17]



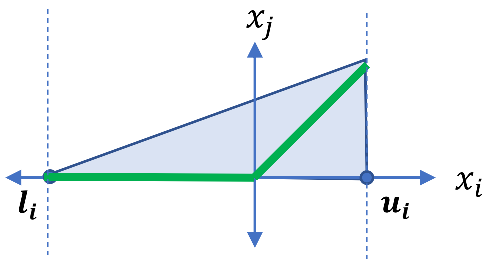
Box [Gehr el al. S&P’18]



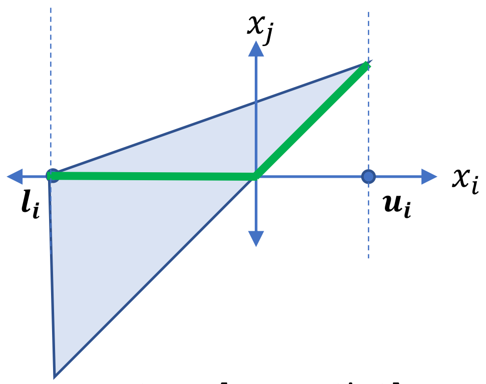
Triangle [Ehlers et al. ATVA’17]



Zonotope:
Wong et al. [ICML’18]
FastLin [ICML’18]
DeepZ [NeurIPS’18]



CROWN [NeurIPS’18]
DeepPoly [POPL’19]



CROWN [NeurIPS’18]
DeepPoly [POPL’19]

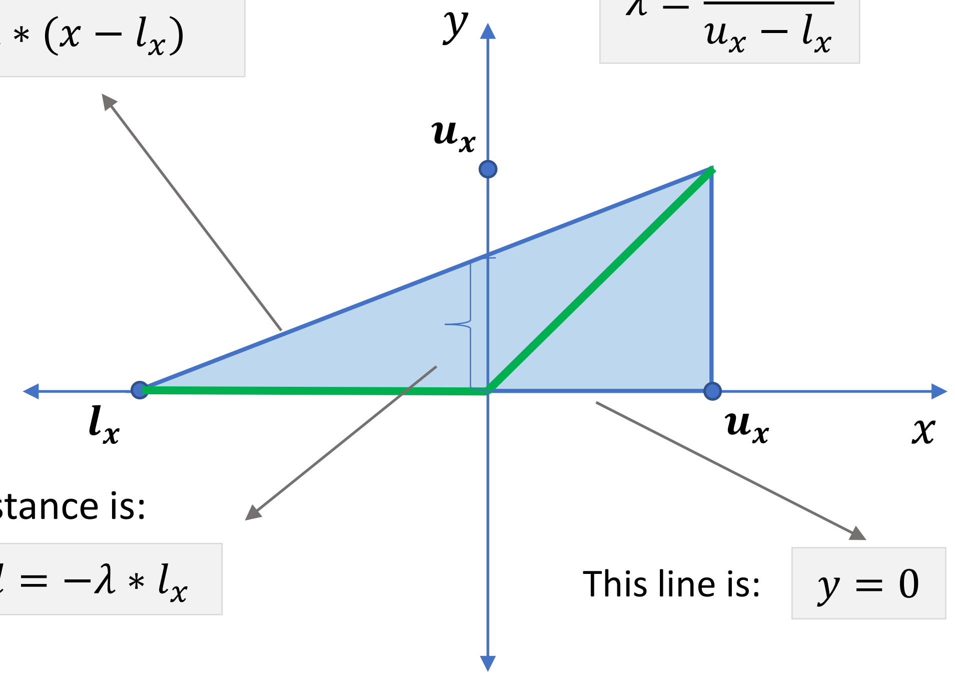
DeepPoly ReLU Relaxaton I

This line is:

$$y = \lambda * x - \lambda * l_x$$
$$= \lambda * (x - l_x)$$

The slope is:

$$\lambda = \frac{u_x}{u_x - l_x}$$



distance is:

$$d = -\lambda * l_x$$

This line is: $y = 0$

So the blue area approximating the green line is captured by 2 inequality constraints below:

$$y \leq \lambda * (x - l_x)$$
$$y \geq 0$$

as well as the upper bound constraint kept as part of the previous layer:

$$x \leq u_x$$

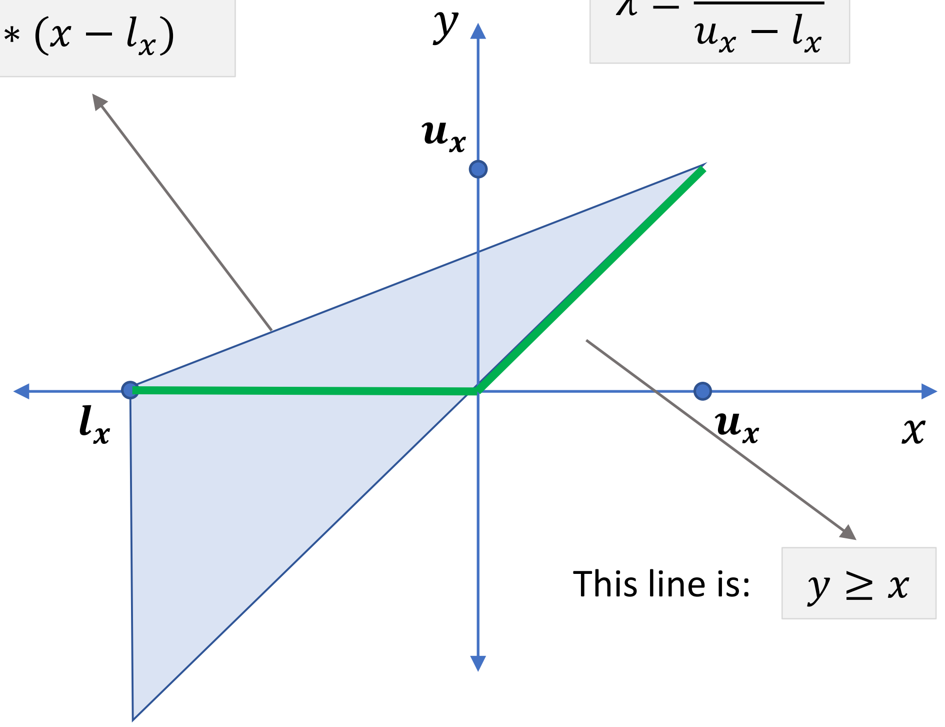
DeepPoly ReLU Relaxaton II

This line is (the same as before):

$$y = \lambda * x - \lambda * l_x$$
$$= \lambda * (x - l_x)$$

The slope is:

$$\lambda = \frac{u_x}{u_x - l_x}$$



This line is: $y \geq x$

So the blue area approximating the green line is captured by 2 inequality constraints below:

$$y \leq \lambda * (x - l_x)$$
$$y \geq x$$

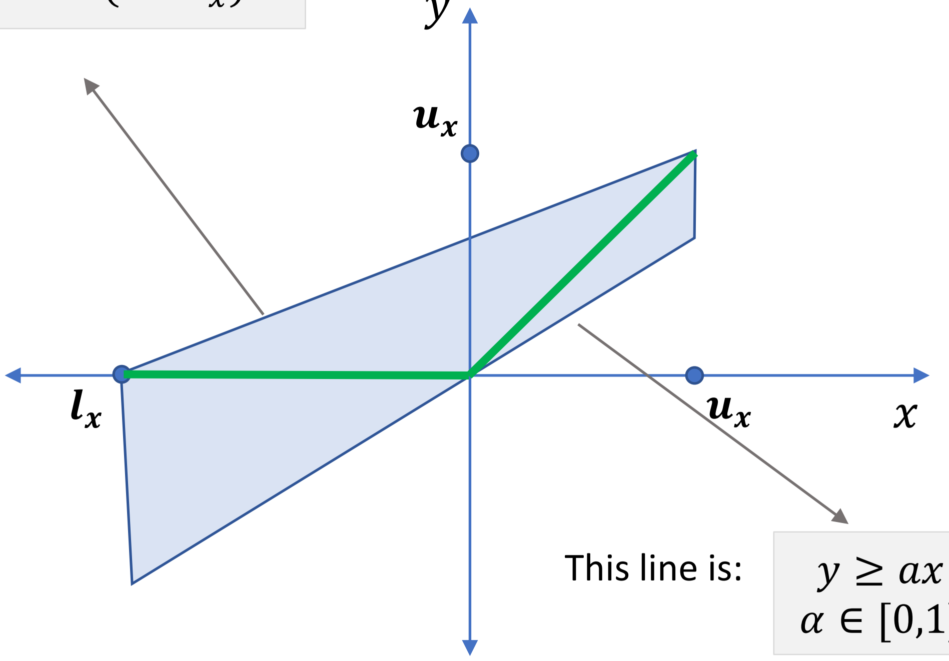
as well as the upper bound constraint kept as part of the previous layer:

$$x \leq u_x$$

DeepPoly ReLU Relaxation α

This line stays the same:

$$y = \lambda * x - \lambda * l_x = \lambda * (x - l_x)$$



This line is:

$$y \geq ax \quad \alpha \in [0,1]$$

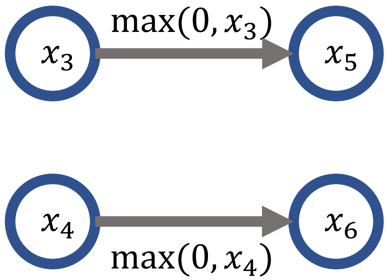
We can **vary the lower bound** constraint $y \geq \alpha x$, $\alpha \in [0,1]$ to get different non-comparable relaxations.

We can **even learn α** to get one more likely to verify the property

(**note**: useful for the project)

Let us apply ReLU relaxation I

$x_3 \geq x_1 + x_2,$
$x_3 \leq x_1 + x_2,$
$l_3 = -2,$
$u_3 = 2$



$x_4 \geq x_1 - x_2,$
$x_4 \leq x_1 - x_2,$
$l_4 = -2,$
$u_4 = 2$

$x_5 \leq \lambda * (x_3 - l_{x3})$
$x_5 \geq 0$

becomes

$x_5 \leq 0.5 * (x_3 + 2)$
$x_5 \geq 0$

$x_6 \leq \lambda * (x_4 - l_{x4})$
$x_6 \geq 0$

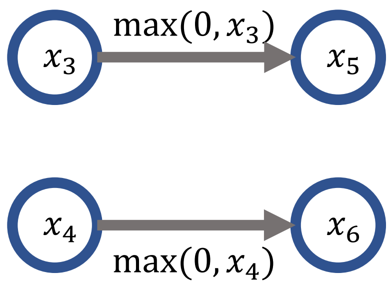
becomes

$x_6 \leq 0.5 * (x_3 + 2)$
$x_6 \geq 0$

Let us apply ReLU relaxation I

$$\begin{gathered} x_3 \geq x_1 + x_2, \\ x_3 \leq x_1 + x_2, \\ l_3 = -2, \\ u_3 = 2 \end{gathered}$$

$$\begin{gathered} x_5 \geq 0, \\ x_5 \leq 0.5 \cdot x_3 + 1, \\ l_5 = 0, \\ u_5 = 2 \end{gathered}$$



we get these constraints after simplification

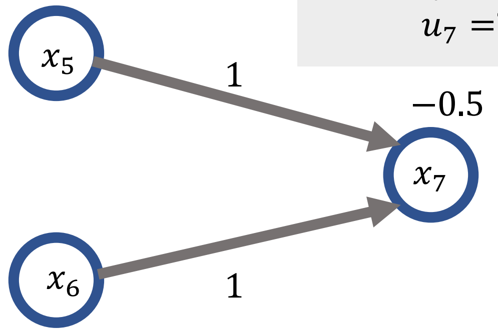
$$\begin{gathered} x_4 \geq x_1 - x_2, \\ x_4 \leq x_1 - x_2, \\ l_4 = -2, \\ u_4 = 2 \end{gathered}$$

$$\begin{gathered} x_6 \geq 0, \\ x_6 \leq 0.5 \cdot x_4 + 1, \\ l_6 = 0, \\ u_6 = 2 \end{gathered}$$

Affine transformation after ReLU

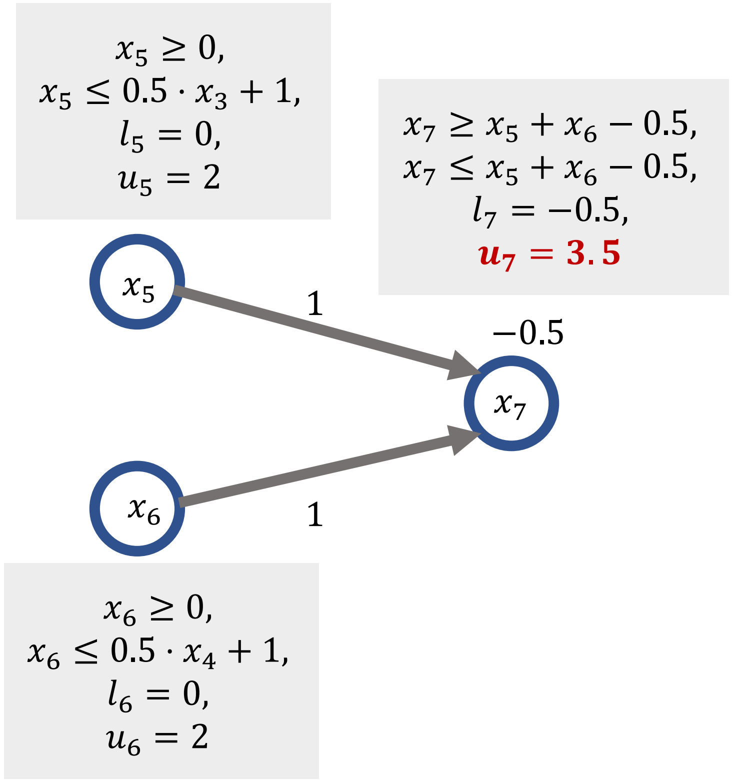
$x_5 \geq 0,$
$x_5 \leq 0.5 \cdot x_3 + 1,$
$l_5 = 0,$
$u_5 = 2$

$x_7 \geq x_5 + x_6 - 0.5,$
$x_7 \leq x_5 + x_6 - 0.5,$
$l_7 =?,$
$u_7 =?$



$x_6 \geq 0,$
$x_6 \leq 0.5 \cdot x_4 + 1,$
$l_6 = 0,$
$u_6 = 2$

Affine transformation after ReLU



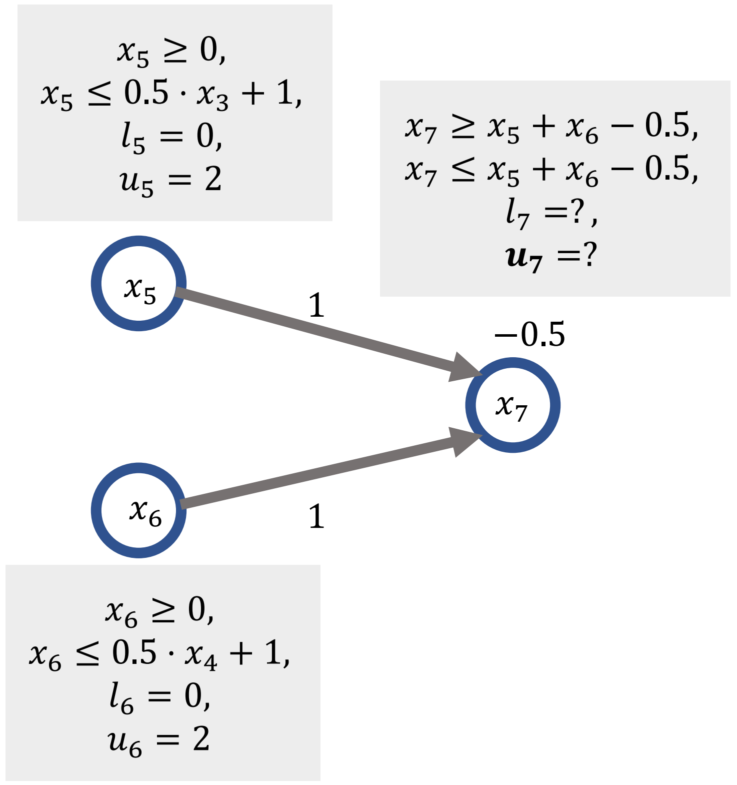
Computing upper bound for neuron x_7 by using upper bounds for u_5 and u_6:

$$x_7 \leq x_5 + x_6 - 0.5$$
$$x_7 \leq 2 + 2 \ - 0.5$$
$$x_7 \leq 3.5$$

Imprecise upper bound u_7 by substituting u_5, u_6 for x_5 and x_6

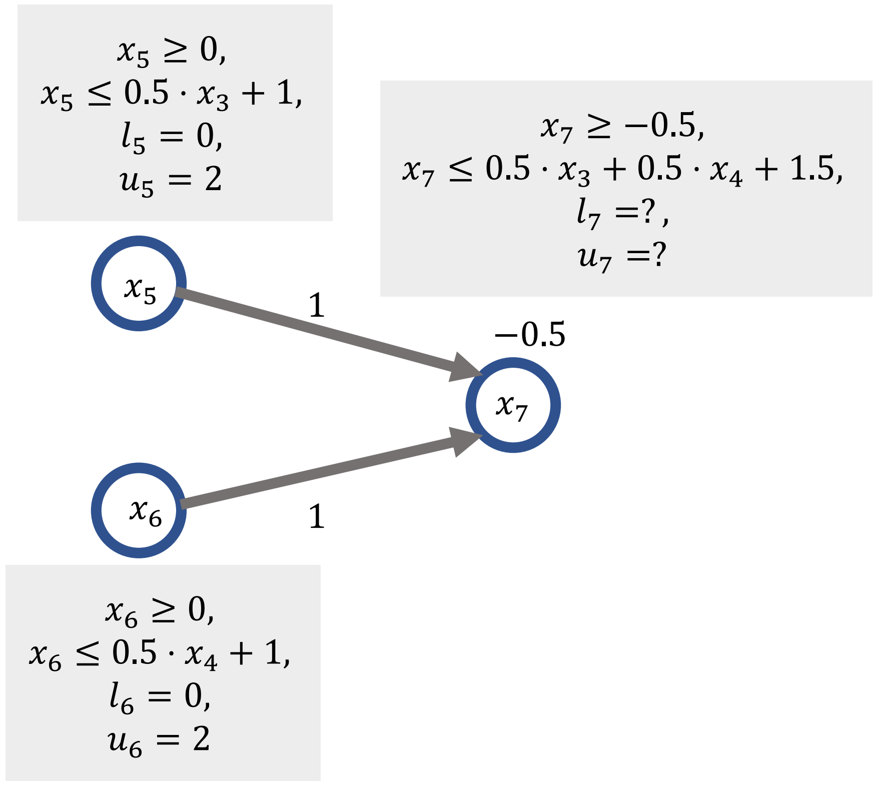
We are trying to compute **sound and precise/tight lower and upper bounds**. To do so, instead of only using the concrete l and u bounds from the previous layer, we can use the relational constraints accumulated in the prior layers. We accomplish this via **backsubstitution**.

Backsubstitution



Obtain the lower and upper bounds for x_7 using the ones from the previous layer

Backsubstitution



Obtain the lower and upper bounds for x_7 using the ones from the previous layer

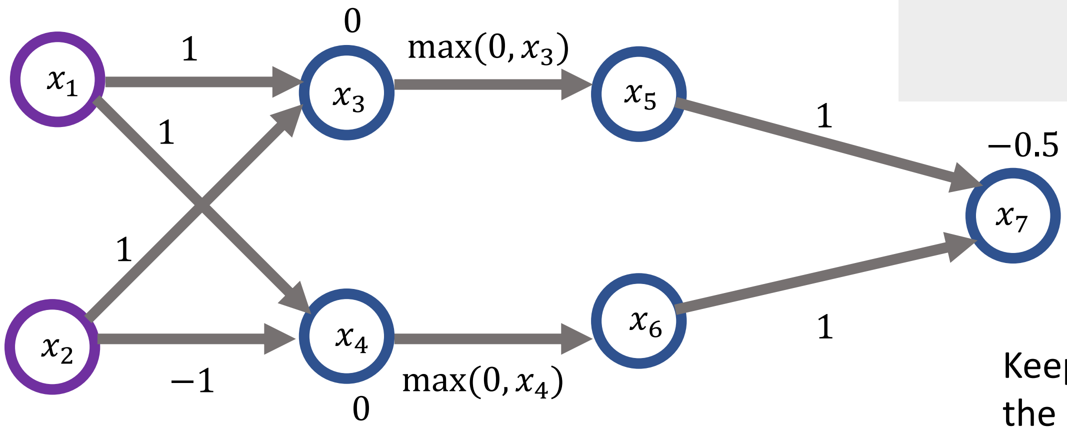
Backsubstitution

$x_1 \geq -1,$
$x_1 \leq 1,$
$l_1 = -1,$
$u_1 = 1$

$x_3 \geq x_1 + x_2,$
$x_3 \leq x_1 + x_2,$
$l_3 = -2,$
$u_3 = 2$

$x_5 \geq 0,$
$x_5 \leq 0.5 \cdot x_3 + 1,$
$l_5 = 0,$
$u_5 = 2$

$x_7 \geq -0.5,$
$x_7 \leq 0.5 \cdot x_3 + 0.5 \cdot x_4 + 1.5,$
$l_7 = ?,$
$u_7 = ?$



$x_2 \geq -1,$
$x_2 \leq 1,$
$l_2 = -1,$
$u_2 = 1$

$x_4 \geq x_1 - x_2,$
$x_4 \leq x_1 - x_2,$
$l_4 = -2,$
$u_4 = 2$

$x_6 \geq 0,$
$x_6 \leq 0.5 \cdot x_4 + 1,$
$l_6 = 0,$
$u_6 = 2$

Keep replacing the bounds for x_7 using the ones from the previous layer:

$x_7 \leq 0.5x_3 + 0.5x_4 + 1.5$
$x_7 \leq 0.5(x_1 + x_2) + 0.5(x_1 - x_2) + 1.5$
$x_7 \leq x_1 + 1.5$
$x_7 \leq 2.5$

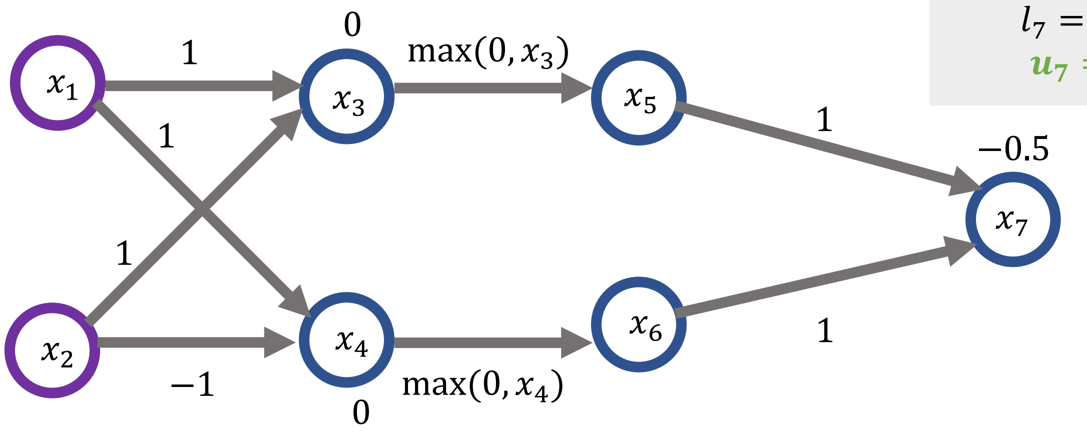
Backsubstitution

$x_1 \geq -1,$
$x_1 \leq 1,$
$l_1 = -1,$
$u_1 = 1$

$x_3 \geq x_1 + x_2,$
$x_3 \leq x_1 + x_2,$
$l_3 = -2,$
$u_3 = 2$

$x_5 \geq 0,$
$x_5 \leq 0.5 \cdot x_3 + 1,$
$l_5 = 0,$
$u_5 = 2$

$x_7 \geq -0.5,$
$x_7 \leq x_1 + 1.5,$
$l_7 = -0.5,$
$\mathbf{u_7 = 2.5}$



$x_2 \geq -1,$
$x_2 \leq 1,$
$l_2 = -1,$
$u_2 = 1$

$x_4 \geq x_1 - x_2,$
$x_4 \leq x_1 - x_2,$
$l_4 = -2,$
$u_4 = 2$

$x_6 \geq 0,$
$x_6 \leq 0.5 \cdot x_4 + 1,$
$l_6 = 0,$
$u_6 = 2$

Dealing with arbitrary input regions

Reminder: in our backsubstitution, we obtained the following steps:

$$x_7 \leq 0.5x_3 + 0.5x_4 + 1.5$$

$$x_7 \leq 0.5(x_1 + x_2) + 0.5(x_1 - x_2) + 1.5$$

$$x_7 \leq x_1 + 1.5$$

$$x_7 \leq 2.5$$

But this step was possible in general as we were dealing with a very specific input region (where each neuron lies in an interval, in this case [-1,1]).

How can we deal with any norm? [Next lecture]

$x_1 \geq -1,$
$x_1 \leq 1,$
$l_1 = -1,$
$u_1 = 1$

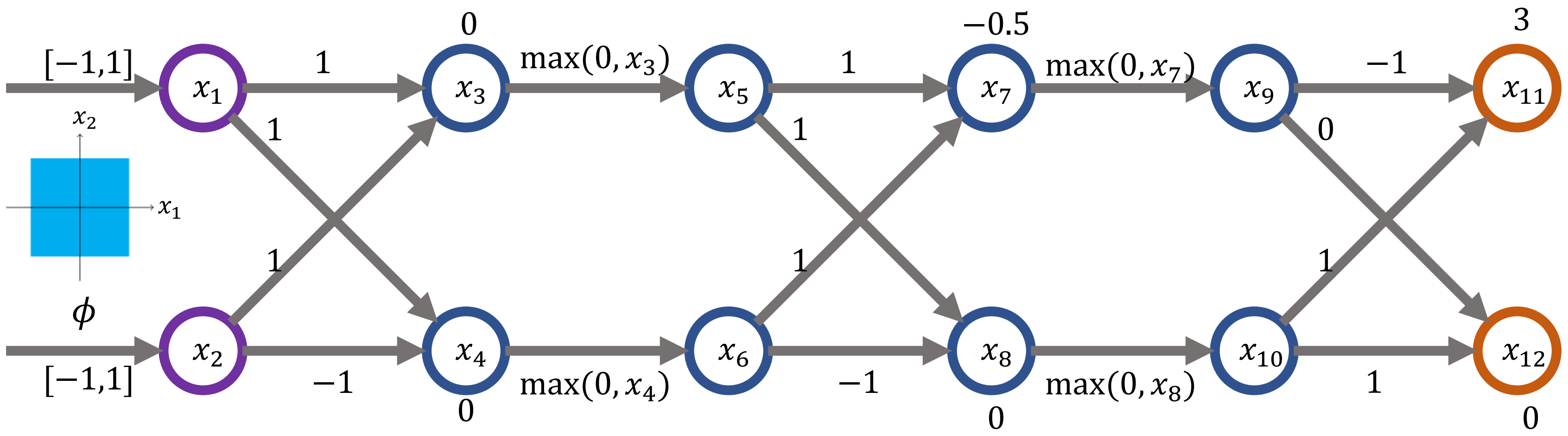
$x_3 \geq x_1 + x_2,$
$x_3 \leq x_1 + x_2,$
$l_3 = -2,$
$u_3 = 2$

$x_5 \geq 0,$
$x_5 \leq 0.5 \cdot x_3 + 1,$
$l_5 = 0,$
$u_5 = 2$

$x_7 \geq x_5 + x_6 - 0.5,$
$x_7 \leq x_5 + x_6 - 0.5,$
$l_7 = -0.5,$
$u_7 = 2.5$

$x_9 \geq 0,$
$x_9 \leq \frac{5}{6} \cdot x_7 + \frac{5}{12}$
$l_9 = 0,$
$u_9 = 2.5$

$x_{11} \geq -x_9 + x_{10} + 3,$
$x_{11} \leq -x_9 + x_{10} + 3,$
$l_{11} = 0.5,$
$u_{11} = 5$



$x_2 \geq -1,$
$x_2 \leq 1,$
$l_2 = -1,$
$u_2 = 1$

$x_4 \geq x_1 - x_2,$
$x_4 \leq x_1 - x_2,$
$l_4 = -2,$
$u_4 = 2$

$x_6 \geq 0,$
x_6
$\leq 0.5 \cdot x_4 + 1,$
$l_6 = 0,$
$u_6 = 2$

$x_8 \geq x_5 - x_6,$
$x_8 \leq x_5 - x_6,$
$l_8 = -2,$
$u_8 = 2$

$x_{10} \geq 0,$
$x_{10} \leq 0.5 \cdot x_8 + 1,$
$l_{10} = 0,$
$u_{10} = 2$

$x_{12} \geq x_{10},$
$x_{12} \leq x_{10},$
$l_{12} = 0,$
$u_{12} = 2$

Proving the robustness property

Goal: Prove $x_{11} - x_{12} > 0$ for all inputs in $[-1,1] \times [-1,1]$

$$x_{11} \geq -x_9 + x_{10} + 3,$$
$$x_{11} \leq -x_9 + x_{10} + 3,$$
$$l_{11} = 0.5,$$
$$u_{11} = 5$$

$$x_{12} \geq x_{10},$$
$$x_{12} \leq x_{10},$$
$$l_{12} = 0,$$
$$u_{12} = 2$$

Proving the robustness property

Goal: Prove $x_{11} - x_{12} > 0$ for all inputs in $[-1,1] \times [-1,1]$

$$\begin{gathered} x_{11} \geq -x_9 + x_{10} + 3, \\ x_{11} \leq -x_9 + x_{10} + 3, \\ l_{11} = 0.5, \\ u_{11} = 5 \end{gathered}$$

$$\begin{gathered} x_{12} \geq x_{10}, \\ x_{12} \leq x_{10}, \\ l_{12} = 0, \\ u_{12} = 2 \end{gathered}$$

Computing lower bound for $x_{11} - x_{12}$ using l_{11}, u_{12} gives -1.5 which is an imprecise result

Proving the robustness property

Goal: Prove $x_{11} - x_{12} > 0$ for all inputs in $[-1,1] \times [-1,1]$

$$\begin{gathered} x_{11} \geq -x_9 + x_{10} + 3, \\ x_{11} \leq -x_9 + x_{10} + 3, \\ l_{11} = 0.5, \\ u_{11} = 5 \end{gathered}$$

$$\begin{gathered} x_{12} \geq x_{10}, \\ x_{12} \leq x_{10}, \\ l_{12} = 0, \\ u_{12} = 2 \end{gathered}$$

Computing lower bound for $x_{11} - x_{12}$ using l_{11}, u_{12} gives -1.5 which is an imprecise result

We now backsubstitute $x_{11} - x_{12}$, and get 0.5, proving the property

DeepPoly Asymptotic Complexity

n: max #neurons in a layer, L: #layers

The **overall asymptotic complexity** is $\mathrm{O}(n^3L^2)$ broken down as follows:

- $\mathrm{O}(n^2L)$: for every neuron, the complexity of doing full back substitution.
- $O(nL)$: is the total number of neurons for which the above takes place.

Note: $O(nL)$ - this is the complexity of all ReLU transformers which are dominated by the above complexity of backsubstitution.

Note: overall complexity of Box is $\mathrm{O}(n^2L)$--$\mathrm{O}(n^2)$ for affine transformation repeated L times.

Summary of Lecture

- We showed how to create a MILP instance for solving the problem of (complete) neural network certification.

- This is useful as MILP alone is prohibitively expensive. This was an example of using an incomplete method to speed up a complete method.

- We discussed one combination of combining complete and incomplete methods. There are other ways for both methods to benefit from each other, we mention some of these in later lectures.

- We defined the DeepPoly convex relaxation, a more precise version of Box, also an instance of an incomplete method.