

Verifying Robustness with Neural Polyhedron Abstraction

Konkuk University

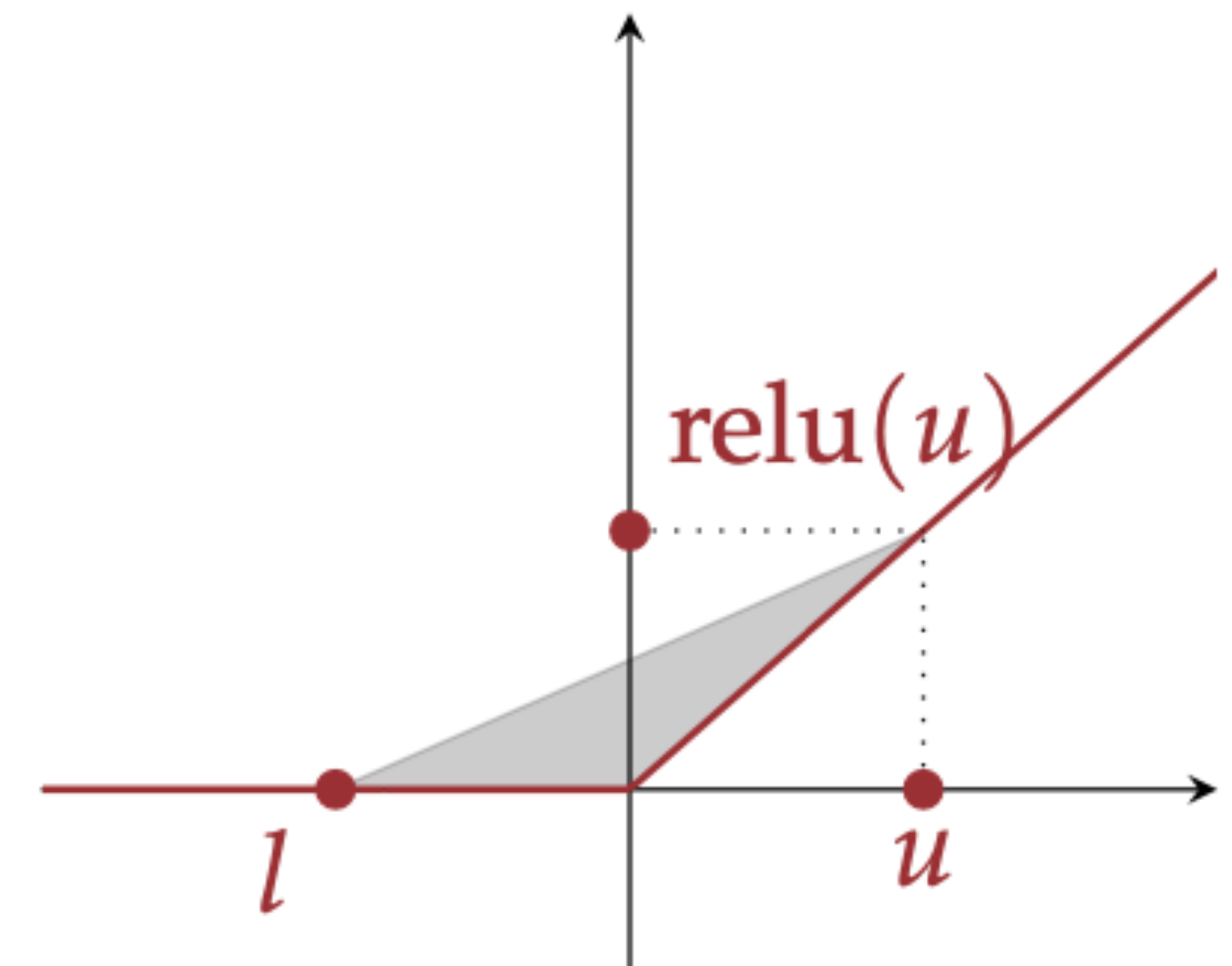
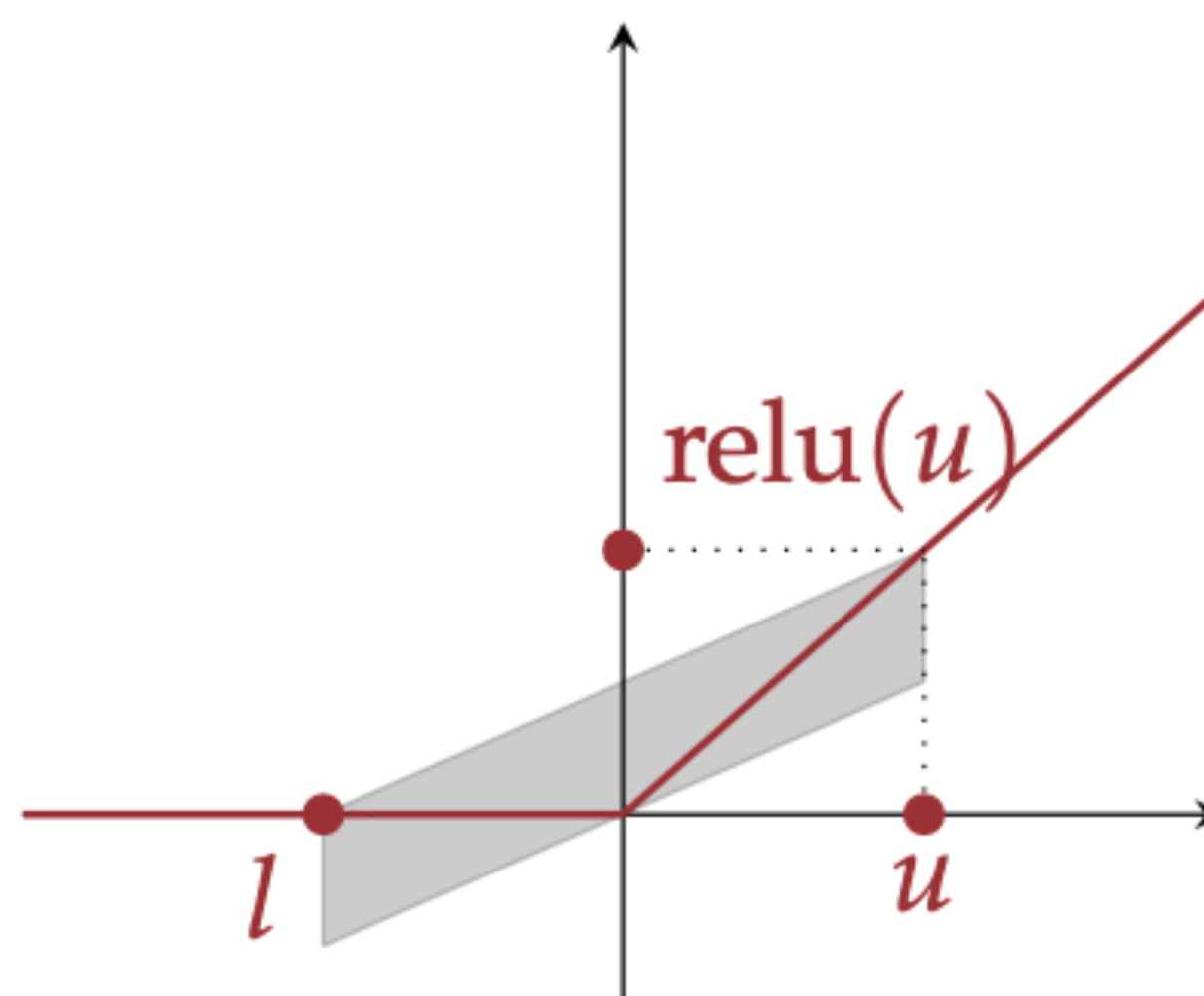
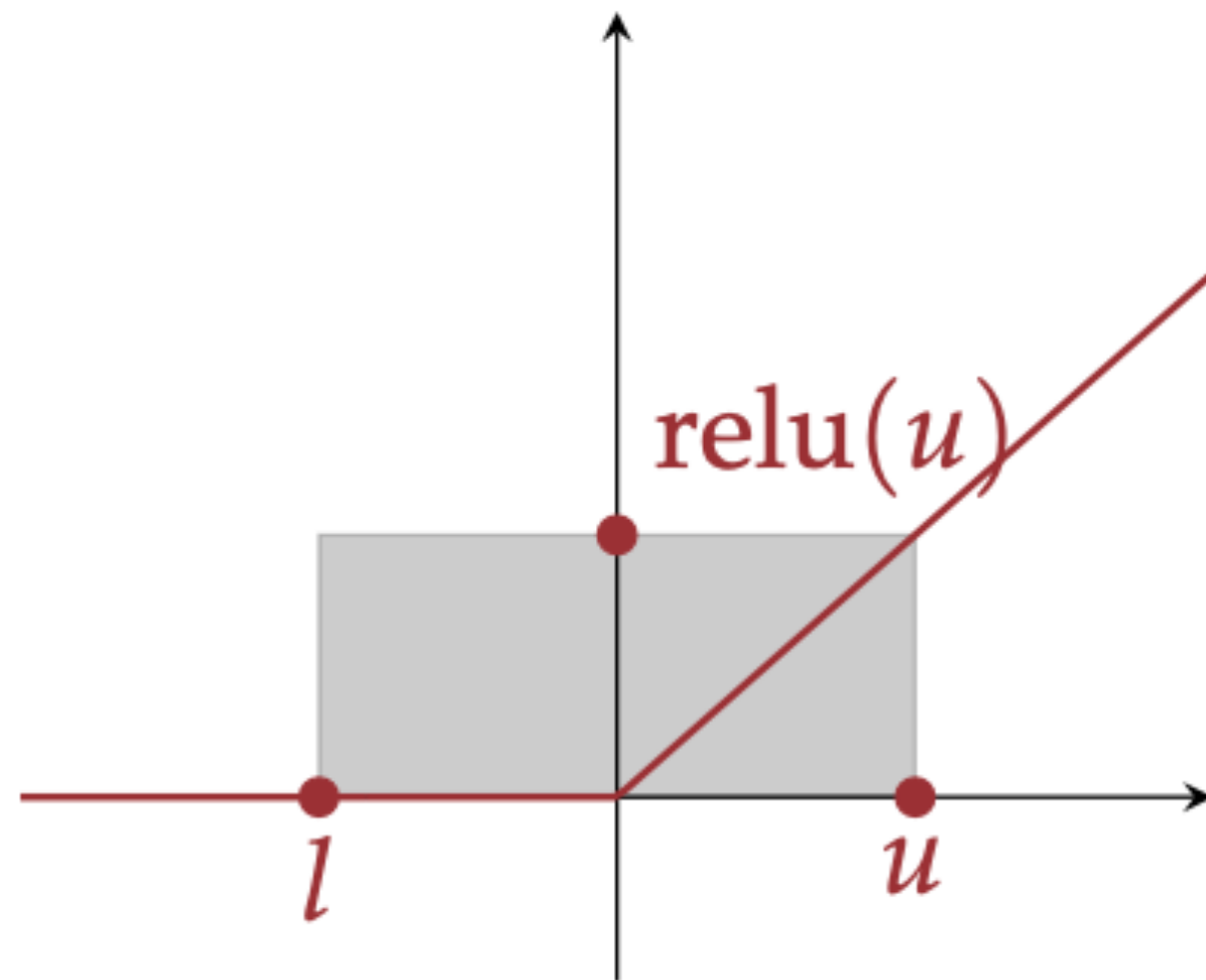
Dept. of Computer Science & Engineering

Kunha Kim

2023 / 10 / 02

Neural Polyhedron Abstraction

Compare with Previous Abstract Domain

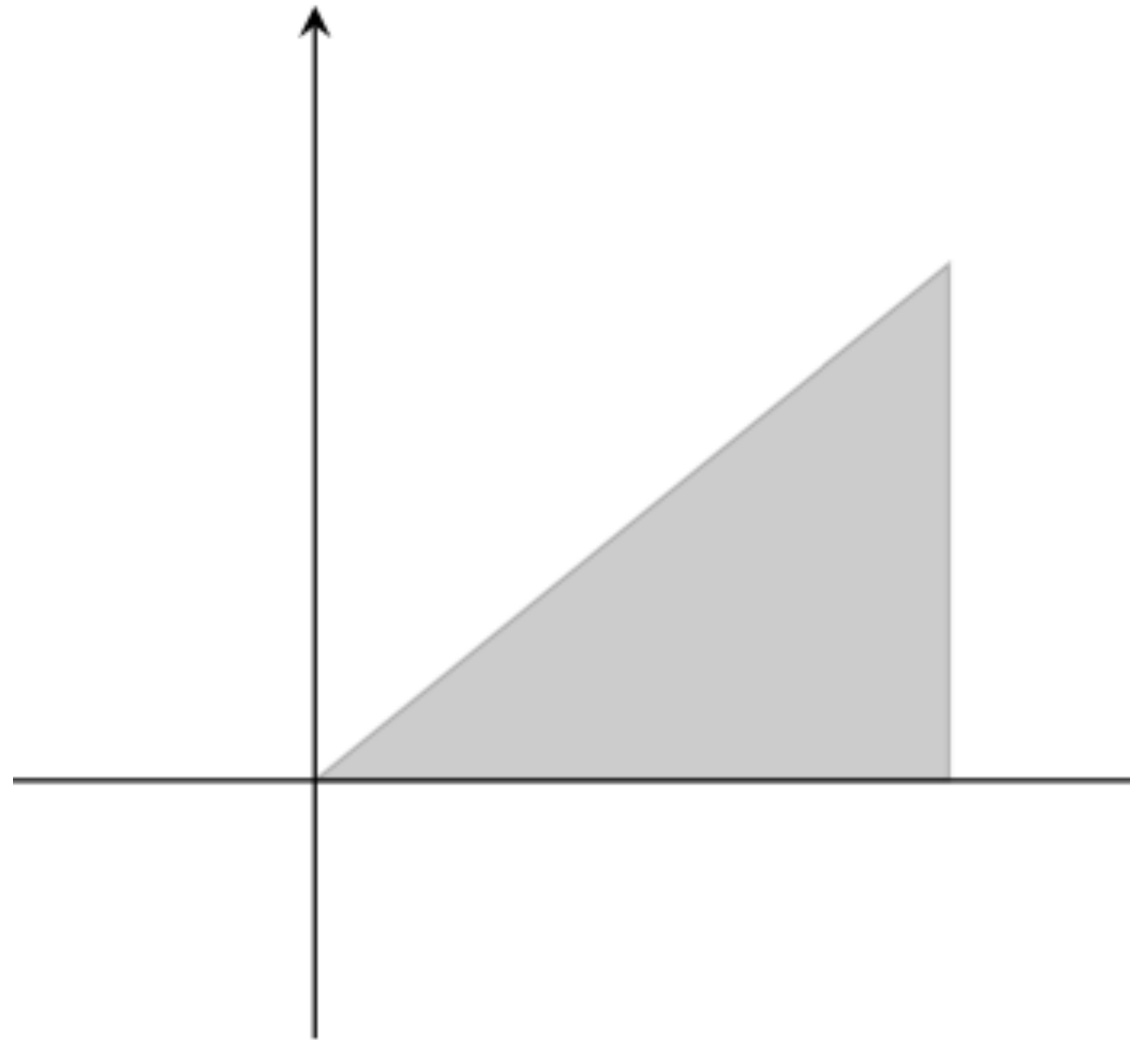


Convex Polyhedra

- We will define a polyhedron in a manner analogous to a zonotope, using a set of m generator variables, $\epsilon_1, \dots, \epsilon_m$.
- With zonotopes the generators are bounded in the interval $[-1, 1]$; with polyhedra, generators are bounded by a set of linear inequalities.
- A zonotope in \mathbb{R}^n is a set of points defined as follows :
 $\{c_{10} + \sum_{i=1}^m c_{1i} \cdot \epsilon_i, \dots, c_{n0} + \sum_{i=1}^m c_{ni} \cdot \epsilon_i \mid F(\epsilon_1, \dots, \epsilon_m)\}$ where F is a boolean function that evaluates to true iff all of its arguments are between -1 and 1.
- With polyhedra, we will define F as a set of linear inequalities over the generator variables, e.g. , $0 \leq \epsilon_1 \leq 5 \wedge \epsilon_1 = \epsilon_2$.

Example

- Consider the following 2-dimensional polyhedron :
 $\{(\epsilon_1, \epsilon_2) \mid F(\epsilon_1, \epsilon_2)\}$ where
 $F \equiv 0 \leq \epsilon_1 \leq 1 \wedge \epsilon_2 \leq \epsilon_1 \wedge \epsilon_2 \geq 0$
- Clearly, this shape is not a zonotope, because its faces are not parallel.

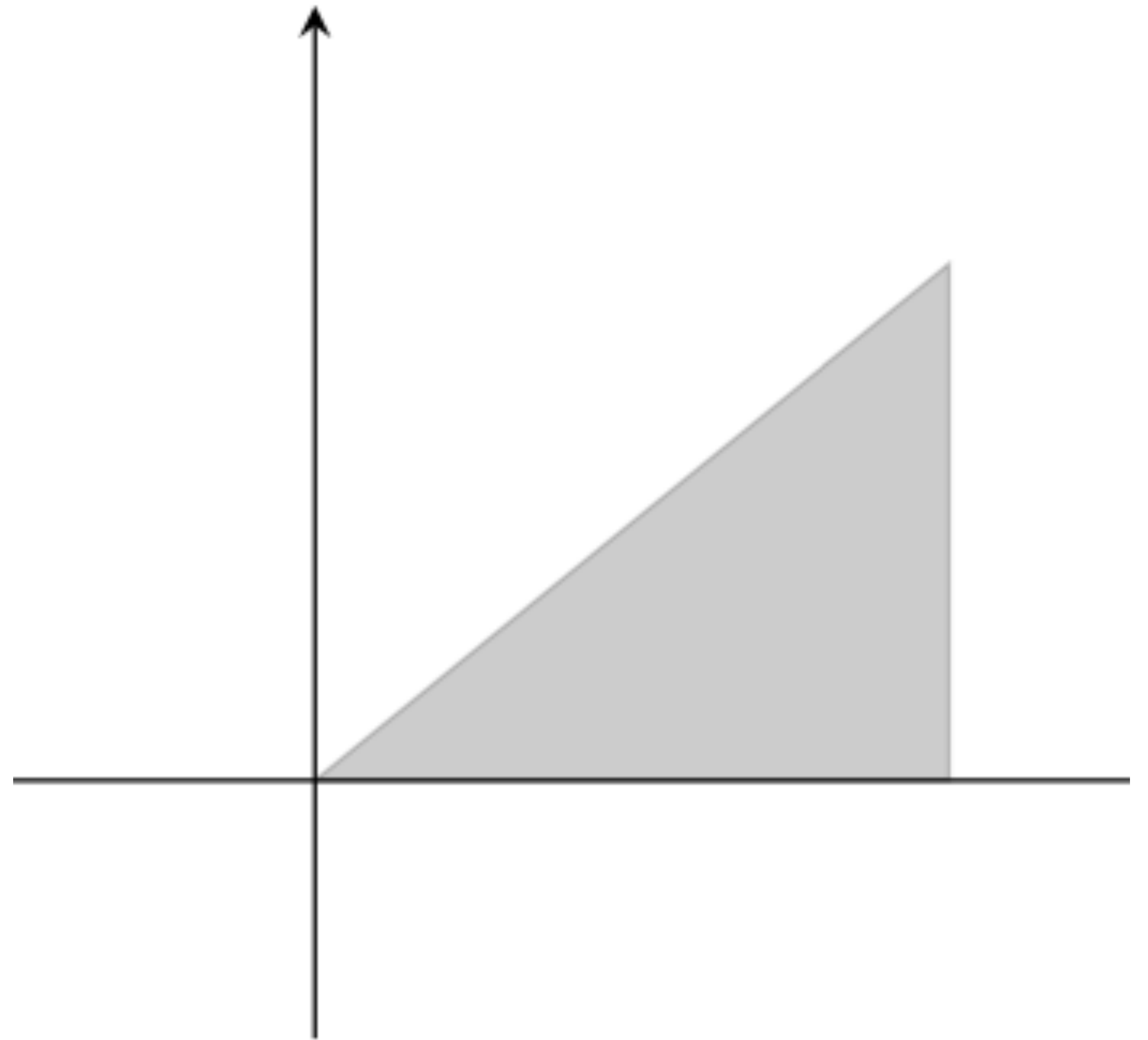


Computing Upper and Lower Bounds

- From now on, given a polyhedron $\{c_{10} + \sum_{i=1}^m c_{1i} \cdot \epsilon_i, \dots, c_{n0} + \sum_{i=1}^m c_{ni} \cdot \epsilon_i \mid F(\epsilon_1, \dots, \epsilon_m)\}$ we will abbreviate it as the tuple : $(\langle c_{1i} \rangle_i, \dots, \langle c_{ni} \rangle_i, F)$.
- Unlike with the interval and zonotope domains, this process is not straightforward.
- Specifically, it involves solving a linear program, which takes polynomial time in the number of variables and constraints.
- To compute the lower bound of the j th dimension, we solve the following linear programming problem: *minimize* $c_{j0} + \sum_{i=1}^m c_{ji} \epsilon_i$ *s . t .* F .
- Similarly, we compute the upper bound of the j th dimension by maximizing instead of minimizing.

Example

- Take our triangle shape from previous example, defined using two generators :
 $(\langle 0,1,0 \rangle, \langle 0,0,1 \rangle, F)$ where
 $F \equiv 0 \leq \epsilon_1 \leq 1 \wedge \epsilon_2 \leq \epsilon_1 \wedge \epsilon_2 \geq 0$
- To compute the upper bound of first dimension, we solve
 $\max \epsilon_1 \text{ s.t. } F.$



Abstract Transformers for Polyhedra : Affine function

- For affine functions, it is really the same transformer as the one for the zonotope domain, except that we carry around the set of linear inequalities F for the zonotope domain, F is fixed throughout.

- Specifically, for the affine function $f(x_1, \dots, x_n) = \sum_j a_j x_j$ where $a_j \in \mathbb{R}$, we

can define the abstract transformer as follows :

$$f^a(\langle c_{1i}, \dots, \langle c_{ni} \rangle, F) = (\langle \sum_j a_j c_{j0}, \dots, \sum_j a_j c_{jm} \rangle, F).$$

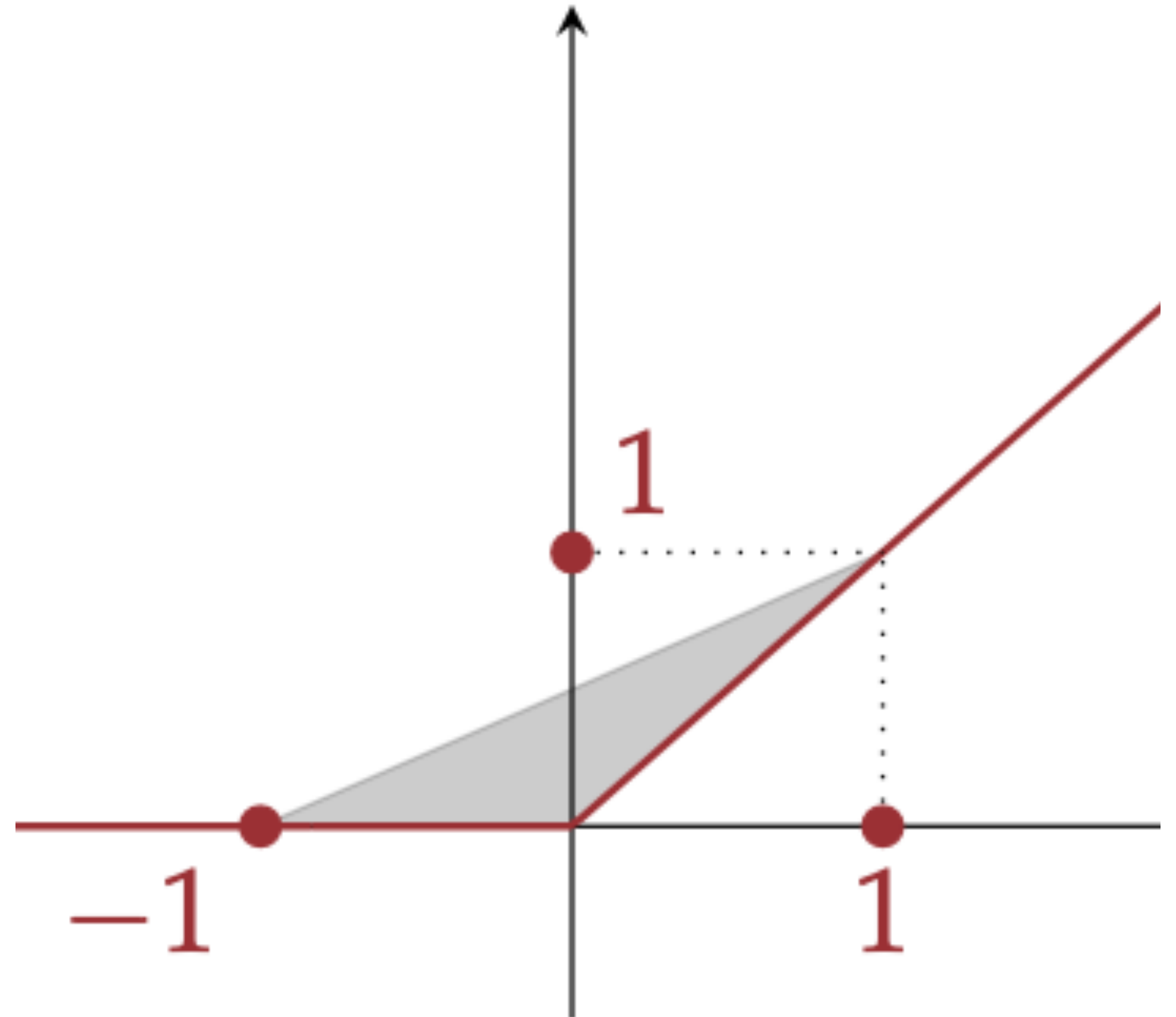
- Consider $f(x, y) = 3x + 2y$. Then, $f^a(\langle 1, 2, 3 \rangle, \langle 0, 1, 1 \rangle, F) = (\langle 3, 8, 11 \rangle, F)$

Abstract Transformer for Polyhedra : ReLU

- The polyhedra which we saw at previous slide is the tightest convex polyhedron we can use to approximate ReLU function.
- The key point is that the top face is the line $y = u(x - l)/u - l$.
- Now, our goal is to define the shaded region, which is bounded by $y = 0$ from below, $y = x$ from the right, and $y = u(x - l)/u - l$ from above.
- This we define $relu^a$ as follows : $relu(\langle c_i \rangle_i, F) = (\langle 0, 0, \dots, 0, 1 \rangle, F')$ where $F' \equiv F \wedge \epsilon_{m+1} \leq u(\langle c_i \rangle - l)/u - l \wedge \epsilon_{m+1} \geq \langle c_i \rangle$.
- $\langle c_i \rangle_i$ is used to denote the full term $c_0 + \sum_{i=1}^m c_i \epsilon_i$.

Example

- Consider $(\langle 0, 1 \rangle, -1 \leq \epsilon_1 \leq 1)$, which is the interval between -1 and 1.
- Invoking $\text{relu}^a(\langle 0, 1 \rangle, -1 \leq \epsilon_1 \leq 1)$ results in $(\langle 0, 0, 1 \rangle, F')$, where
$$F' \equiv -1 \leq \epsilon_1 \leq 1 \wedge \epsilon_2 \leq (\epsilon_1 + 1)/2 \wedge \epsilon_2 \geq 0 \wedge \epsilon_2 \geq \epsilon_1$$
- The shaded region defined by F' is at right.



Abstractly Interpreting Neural Networks with Polyhedra

- Recall that a neural network is defined as a graph $G = (V, E)$, giving rise to a function $f_G : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where $n = |V^{in}|$ and $m = |V^{out}|$. We define $f_G^a(\langle c_{1j} \rangle, \dots, \langle c_{nj} \rangle, F)$ as follows
- First, for every input node v_i , we define $out^a(v_i) = (\langle c_{ij} \rangle_j, F)$.
- Second, for every non-input node v , we define $out^a(v) = f_v^a(p_1, \dots, p_k, \bigwedge_{i=1}^k F_i)$ where f_v^a is the abstract transformer of f_v , v has incoming edges $(v_1, v), \dots, (v_k, v)$, and $out^a(v_i) = (p_i, F_i)$.
- Observe what is happening here: we're combining (with \wedge) the constraints from the incoming edges. This ensures that we capture the relations between incoming values.
- Finally, the output of f_G^a is the m -dimensional polyhedron $(p_1, \dots, p_m, \bigwedge_{i=1}^m F_i)$ where v_1, \dots, v_m are the output nodes and $out^a(v_i) = (p_i, F_i)$.

An Abstract Domain for Certifying Neural Networks : Singh et al.

Introduction

- Over the last few years, deep neural networks have become increasingly popular and have now started penetrating safety critical domains such as autonomous driving [Bojarski et al. 2016] and medical diagnosis [Amato et al. 2013] where they are often relied upon for making important decisions.
- As a result of this widespread adoption, it has become even more important to ensure that neural networks behave reliably and as expected.
- Unfortunately, reasoning about these systems is challenging due to their “black box” nature: it is difficult to understand what the network does since it is typically parameterized with thousands or millions of real-valued weights that are hard to interpret.

Introduction(Cont.)

- To address the challenge of reasoning about neural networks, recent research has started exploring new methods and systems which can automatically prove that a given network satisfies a specific property of interest (e.g., robustness to certain perturbations, pre/post conditions).
- Despite the progress made by previous works, more research is needed to reach the point where we are able to solve the overall neural network reasoning challenge successfully.
- In particular, we still lack an analyzer that can scale to large networks, is able to handle popular neural architectures (e.g., feedforward, convolutional), and yet is sufficiently precise to prove relevant properties required by applications.

Introduction(Cont.)

- In this work, we propose a new method and system, called DeepPoly, that makes a step forward in addressing the challenge of verifying neural networks with respect to both scalability and precision.
- The key technical idea behind DeepPoly is a novel abstract interpreter specifically tailored to the setting of neural networks.
- Concretely, our abstract domain is a combination of floating-point polyhedra with intervals, coupled with abstract transformers for common neural network functions such as affine transforms, the rectified linear unit (ReLU), sigmoid and tanh activations, and the maxpool operator.
- As a result, DeepPoly is more precise than Weng et al. [2018], Gehr et al. [2018] and Singh et al. [2018a], yet can handle large convolutional networks and is sound for floating point arithmetic.







Attack	Original	Lower	Upper
L_∞			
Rotation			

Fig. 1. Two different attacks applied to MNIST images.

Overview : Specification

- Suppose we work with a hypothetical image that contains only two pixels and the perturbation is such that it places both pixels in the range $[-1, 1]$ (pixels are usually in the range $[0, 1]$, however, we use $[-1, 1]$ to better illustrate our analysis).
- Our goal will be to prove that the output of the network at one of the output neurons is always greater than the output at the other one, for any possible input of two pixels in the range $[-1, 1]$.
- If the proof is successful, it implies that the network produces the same classification label for all of these images.

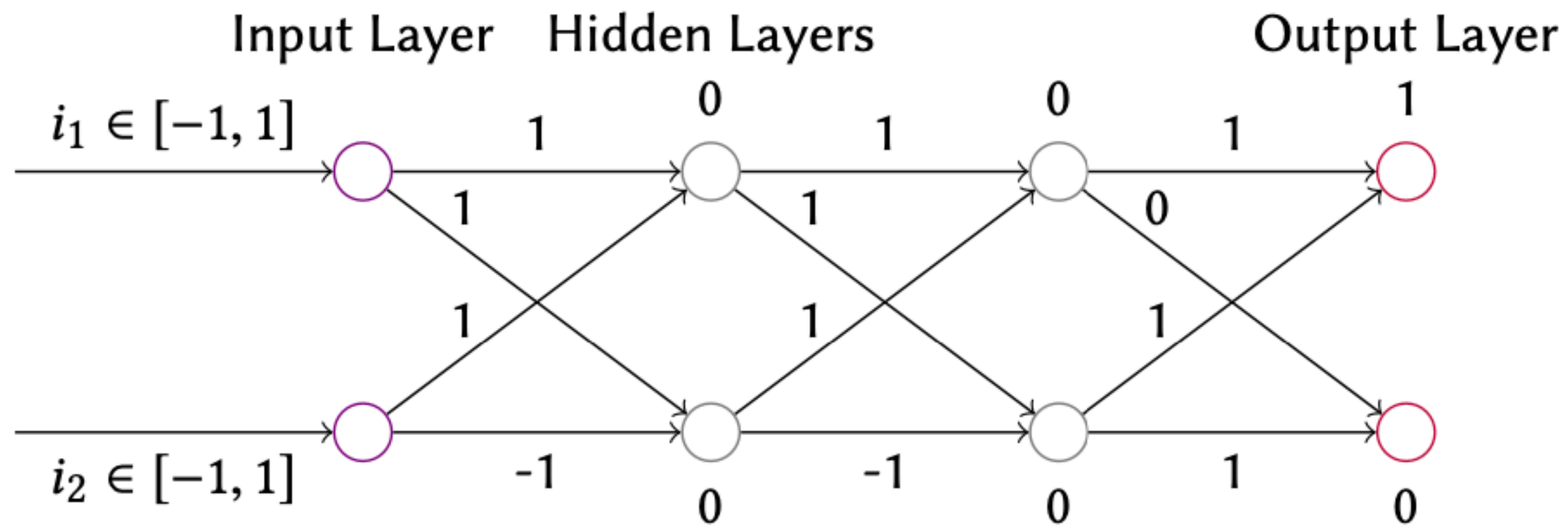


Fig. 2. Example feedforward neural network with ReLU activations.

Overview : Abstract domain

- Our abstract domain, formally described in Section 4, associates two constraints with each variable x_i : an upper polyhedral constraint and a lower polyhedral constraint.
- Additionally, the domain tracks auxiliary (concrete) bounds, one upper bound and one lower bound for each variable, describing a bounding box of the concretization of the abstract element.
- Our domain is less expressive than the Polyhedra domain [Cousot and Halbwachs 1978] because it bounds the number of conjuncts that can appear in the overall formula to $2n$ where n is the number of variables of the network.
- Such careful restrictions are necessary because supporting the full expressive power of convex polyhedra leads to an exponential number of constraints that make the analysis for thousands of neurons practically infeasible.

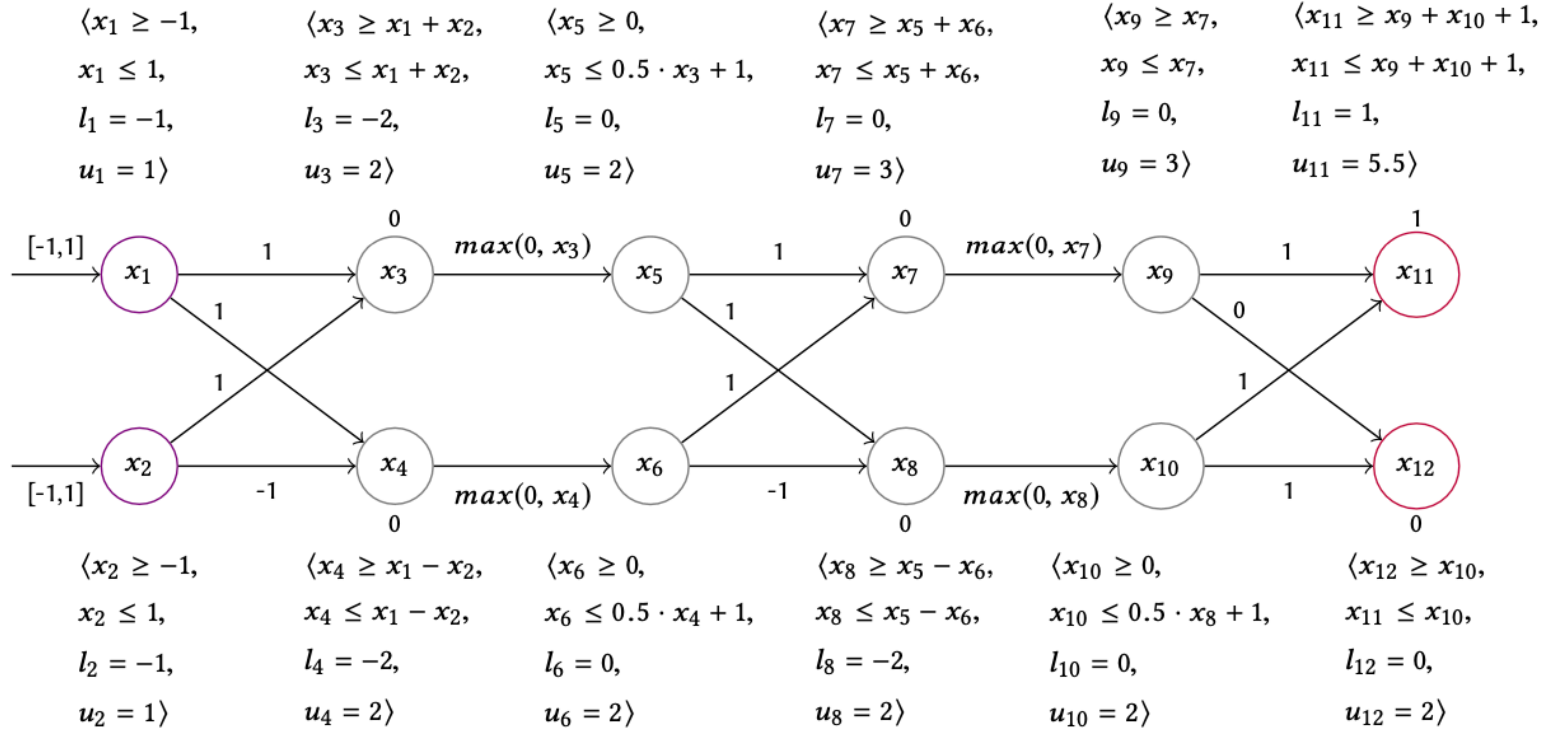


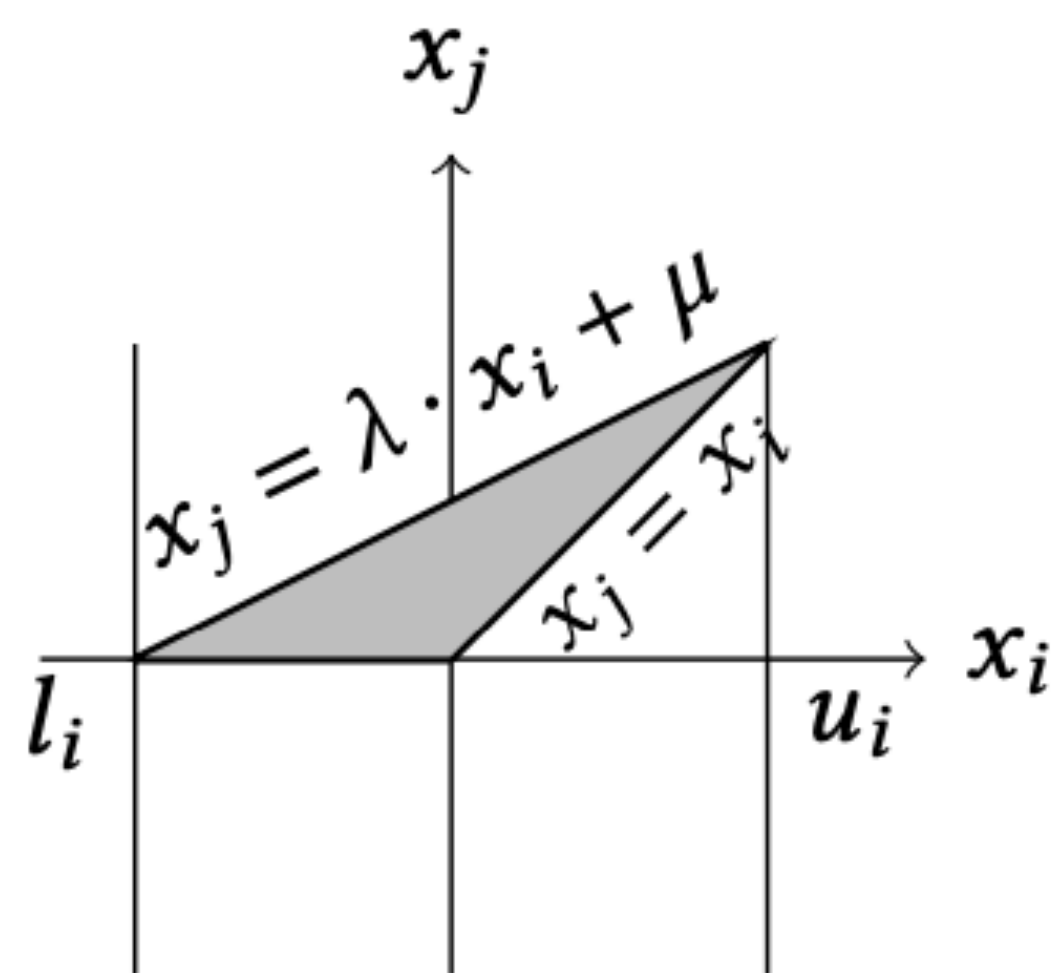
Fig. 3. The neural network from Fig. 2 transformed for analysis with the abstract domain.

Overview : Abstract domain(Cont.)

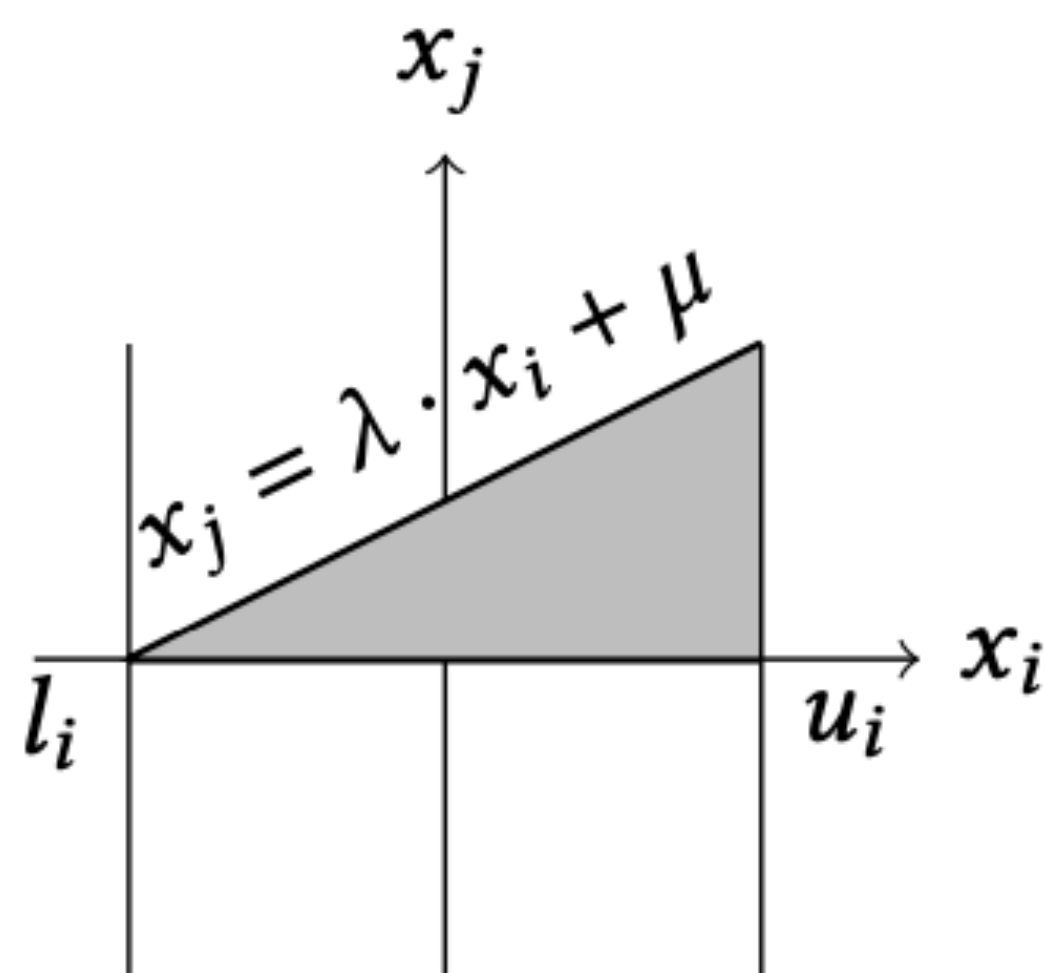
- First, the lower and upper relational polyhedral constraints associated with x_i have the form $v + \sum_j w_j \cdot x_j$ where $v \in \mathbb{R} \cup \{-\infty, \infty\}$, $w \in \mathbb{R}^n$, $\forall j \geq i . w_j = 0$.
- Second, for the concrete lower and upper bounds of x_i , we use $l_i, u_i \in \mathbb{R} \cup \{-\infty, \infty\}$, respectively.
- All abstract elements a in our domain satisfy the additional invariant that the interval $[l_i, u_i]$ overapproximates the set of values that the variable x_i can take.
- Here we start with $x_1 = [-1, 1]$ and $x_2 = [-1, 1]$.

Restriction of polyhedral constraints

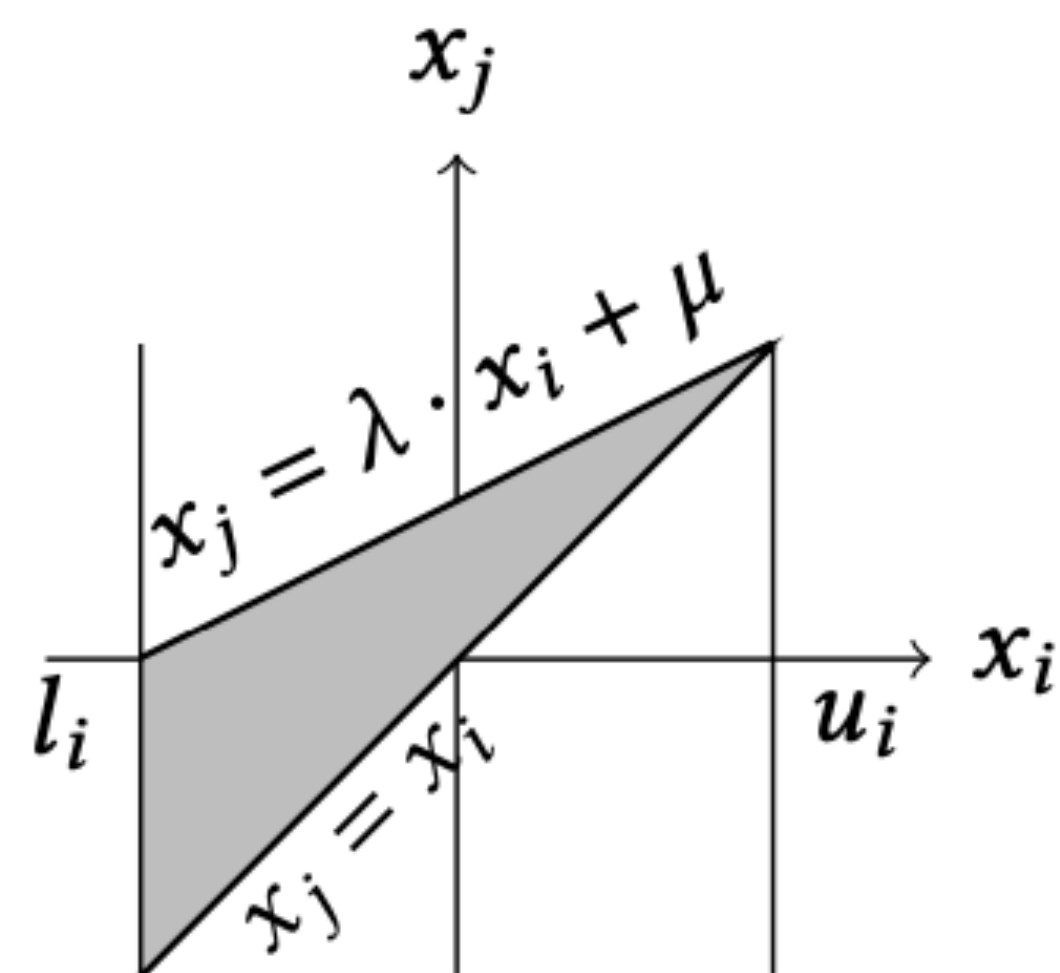
- Next, the affine transform at the first layer updates the constraints for x_3 and x_4 . The abstract transformer first adds the constraints $x_1 + x_2 \leq x_3 \leq x_1 + x_2$, $x_1 - x_2 \leq x_4 \leq x_1 - x_2$.
- For the case $u_i \leq 0$ or $l_i > 0$, we don't have any problems. But when $l_i < 0$ and $u_i > 0$, we have to choose how to loose information.
- The approximation of Fig. 4 (a) minimizes the area in the x_i, x_j plane, and would add the following relational constraints and concrete bounds for x_j : $x_i \leq x_j$, $0 \leq x_j$, $x_j \leq u_i(x_i - l_i)/u_i - l_i$, $l_j = 0$, $u_j = u_i$.
- However, we will not use this because of exponential blowup. The polyhedral constraints must have one upper bound constraints and one lower bound constraints.
- We have two choices : $0 \leq x_j \leq u_i(x_i - l_i)/u_i - l_i$, $l_j = 0$, $u_j = u_i$ or $x_i \leq x_j \leq u_i(x_i - l_i)/u_i - l_i$, $l_j = l_i$, $u_j = u_i$.



(a)



(b)



(c)

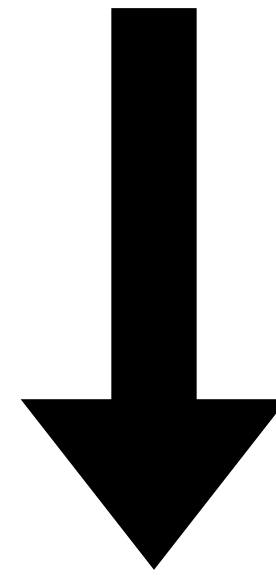
Fig. 4. Convex approximations for the ReLU function: (a) shows the convex approximation with the minimum area in the input-output plane, (b) and (c) show the two convex approximations proposed in this paper. In the figure, $\lambda = u_i/(u_i - l_i)$ and $\mu = -l_i u_i/(u_i - l_i)$.

Restriction of polyhedral constraints(Cont.)

- Note that it would be incorrect to set $l_j = 0$ in (4) above (instead of $l_j = l_i$).
- The reason is that this would break a key domain invariant which we aim to maintain, namely that the concretization of the two symbolic bounds for x_j is contained inside the concretization of the concrete bounds l_j and u_j .
- In particular, if we only consider the two symbolic bounds for x_j , then x_j would be allowed to take on negative values and these negative values would not be included in the region $[0, u_i]$.
- This domain invariant is important to ensure efficiency of our transformers and as we prove later, all of our abstract transformers maintain it.
- We choose tighter approximation, when $u_i \leq -l_i$, we use (b). Otherwise, we use (c).

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

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



$$x_5 + x_6 \leq x_7 \leq x_5 + x_6,$$

$$x_5 - x_6 \leq x_8 \leq x_5 - x_6.$$

Substituting concrete bounds

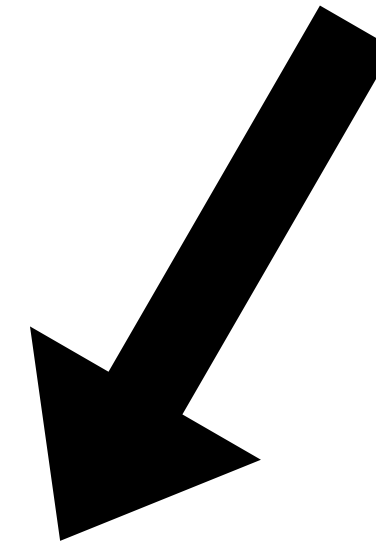
$$x_5 + x_6 \leq x_7 \leq x_5 + x_6,$$

$$x_5 - x_6 \leq x_8 \leq x_5 - x_6.$$



$$0 \leq x_7 \leq 0.5 \cdot x_3 + 0.5 \cdot x_4 + 2,$$

$$-0.5 \cdot x_4 - 1 \leq x_8 \leq 0.5 \cdot x_3 + 1.$$



$$0 \leq x_7 \leq x_1 + 2,$$

$$-0.5 \cdot x_1 + 0.5 \cdot x_2 - 1 \leq x_8 \leq 0.5 \cdot x_1 + 0.5 \cdot x_2 + 1.$$

Avoiding exponential blowup

- Fig. 4 (a)'s approximation causes exponential blowup. In this case, we have two lower relational constraints.
- More generally, if the affine expression for a variable x_i contains p variables with positive coefficients and n variables with negative coefficients, then the number of possible lower and upper relational constraints is 2^p and 2^n , respectively, leading to an exponential blowup.
- Consider x_5 and x_6 with $x_3 \leq x_5$, $0 \leq x_5$, $x_5 \leq u_3(x_3 - l_3)/u_3 - l_3$ and $x_4 \leq x_6$, $0 \leq x_6$, $x_6 \leq u_4(x_4 - l_4)/u_4 - l_4$. And check x_7 with these constraints. See next slide.

Avoiding exponential blowup

- x_7 has four constraints by substituting x_5 and x_6 to x_3 and x_4 .
- $0 \leq x_7 \leq u_3(x_3 - l_3)/(u_3 - l_3) + u_4(x_4 - l_4)/(u_4 - l_4)$
- $x_3 \leq x_7 \leq u_3(x_3 - l_3)/(u_3 - l_3) + u_4(x_4 - l_4)/u_4 - l_4$
- $x_4 \leq x_7 \leq u_3(x_3 - l_3)/(u_3 - l_3) + u_4(x_4 - l_4)/(u_4 - l_4)$
- $x_3 + x_4 \leq x_7 \leq u_3(x_3 - l_3)/u_3 - l_3 + u_4(x_4 - l_4)/(u_4 - l_4)$.

Asymptotic runtime

- The computation of concrete bounds by the abstract affine transformer in the hidden layers is the most expensive step of our analysis.
- If there are L network layers and the maximum number of variables in a layer is n_{max} , then this step for one variable is in $O(n_{max}^2 \cdot L)$.
- Storing the concrete bounds ensures that the subsequent ReLU transformer has constant cost.
- All our transformers work point-wise, i.e., they are independent for different variables since they only read constraints and bounds from the previous layers. This makes it possible to parallelize our analysis on both CPUs and GPUs.

Precision vs Performance trade-off

- We also note that our approach allows one to easily vary the precision-performance knob of the affine transformer in the hidden layers
- We can select a subset of variables for which to perform complete substitution all the way back to the first layer (the example above showed this for all variables).
- Also, we can decide at which layer we would like to stop the substitution and select the concrete bounds at that layer.

Checking Specification

- After some computations, we get $x_9 + x_{10} + 1 \leq x_{11} \leq x_9 + x_{10} + 1$ and $x_{10} \leq x_{12} \leq x_{10}$. With backsubstitution, we get $l_{11} = 1$, $u_{11} = 5.5$ and $l_{12} = 0$, $u_{12} = 2$.
- Formally, our specification is $\forall i_1, i_2 \in [-1, 1] \times [-1, 1], x_{11} > x_{12}$ or $\forall i_1, i_2 \in [-1, 1] \times [-1, 1], x_{11} < x_{12}$.
- But to prove $x_{11} - x_{12} > 0$ or $x_{12} - x_{11} > 0$, it's too imprecise. So we add new variable $x_{13} := x_{11} - x_{12}$ with $x_{11} - x_{12} \leq x_{13} \leq x_{11} - x_{12}$.
- After back substitution, we get $l_{13} = 1$ and $u_{13} = 4$. Thus we have proved our specification.

Abstract Domain and Transformers

- Elements in our abstract domain \mathcal{A}_n consist of a set of polyhedral constraints of a specific form, over n variables.
- Formally, an abstract element $a \in \mathcal{A}_n$ over n variables can be written as a tuple $a = \langle a^{\leq}, a^{\geq}, l, u \rangle$ where $a_i^{\leq}, a_i^{\geq} \in \{x \mapsto v + \sum_{j \in [i-1]} w_j \cdot x_j \mid v \in \mathbb{R} \cup \{-\infty, \infty\}, w \in \mathbb{R}^{i-1}\}$ for $i \in [n]$ and $l, u \in (\mathbb{R} \cup \{-\infty, \infty\})^n$. Here, we use the notation $[n] := \{1, 2, \dots, n\}$.
- The concretization function $\gamma_n : \mathcal{A}_n \rightarrow P(\mathbb{R}^n)$ is then given by $\gamma_n(a) = \{x \in \mathbb{R}^n \mid \forall i \in [n]. a_i^{\leq}(x) \leq x_i \wedge a_i^{\geq}(x) \geq x_i\}$.

Domain Invariant

- All abstract elements in our domain additionally satisfy the following invariant:

$$\gamma_n(a) \subseteq \times_{i \in [n]} [l_i, u_i].$$

- In other words, every abstract element in our domain maintains concrete lower and upper bounds which over-approximate the two symbolic bounds.
- To simplify our exposition of abstract transformers, we will only consider the case where all variables are bounded, which is always the case when our analysis is applied to neural networks.
- Our abstract transformers $T_f^\#$ for a deterministic function $f : \mathcal{A}^m \rightarrow \mathcal{A}^n$ satisfy the following soundness property : $T_f(\gamma_m(a)) \subseteq \gamma_n(T_f^\#(a))$ for all $a \in \mathcal{A}^m$, where T_f is the corresponding concrete transformer of f , given by $T_f(X) = \{f(x) \mid x \in X\}$.

ReLU Abstract Transformer

- Let $f: \mathbb{R}^{i-1} \rightarrow \mathbb{R}^i$ be a function that executes the assignment $x_i \leftarrow \max(0, x_j)$ for some $j < i$.
- The corresponding abstract ReLU transformer is $T_f^\#(\langle a^\leq, a^\geq, l, u \rangle) = \langle a'^\leq, a'^\geq, l', u' \rangle$ where $a'_k{}^\leq = a_k{}^\leq$, $a'_k{}^\geq = a_k{}^\geq$, $l'_k = l_k$ and $u'_k = u_k$ for $k < i$.
- If $u_j \leq 0$, then $a_i'^\leq(x) = a_i'^\geq = 0$. If $0 \leq l_j$, then $a_i'^\leq(x) = a_i'^\geq = x_j$.

ReLU Abstract Transformer

- For the case $l_i < 0$ and $u_i > 0$, the abstract ReLU transformer approximates the assignment by a set of linear constraints forming its convex hull when it is restricted to the interval $[l_j, u_j]$:
$$0 \leq x_i, x_j \leq x_i, x_i \leq u_j(x_j - l_j)/(u_j - l_j).$$
- As there is only one upper bound for x_i , we obtain the following rule : $a_i^{\geq}(x) = u_j(x_j - l_j)/(u_j - l_j)$.
- On the other hand, we have two lower bounds for x_i . Any convex combination of those two constraints is still a valid lower bound. Therefore, we can set $a_i^{\leq}(x) = \lambda \cdot x_j$ for any $\lambda \in [0,1]$.
- We select the $\lambda \in [0,1]$ that minimizes the area of resulting shape in the (x_i, x_j) -plane.
- Finally, we set $l'_i = \lambda \cdot l_j$ and $u'_i = u_j$.