Lecture 9: Verifying Robustness

Trustworthy Machine Learning

Spring 2024

Formal Methods for Verified Robustness

- **Last lecture:**
  - **Formalizing program verification: Pre/post conditions**
  - **Verification as constraint solving**
  - **Robustness checking as program verification**

- **Today:**
  - **Specialized constraint solver ReluPlex for neural network verification**
  - **Verifying robustness by abstract interpretation (box domain)**

Slides credit: Gagandeep Singh and Madhu Parthasarathy (UIUC)

Specifications over DNNs

Precondition

$$\forall x_1, x_2.\, l_1 \le x_1 \le u_1, l_2 \le x_2 \le u_2$$

DNN f

def $f(x_1, x_2)$:
 $x_3 = w_{13} \cdot x_1 + w_{23} \cdot x_2 + b_3$
 $x_4 = w_{14} \cdot x_1 + w_{24} \cdot x_2 + b_4$
 $x_5 = \max(0, x_3)$
 $x_6 = \max(0, x_4)$
 $x_7 = w_{57} \cdot x_5 + w_{67} \cdot x_6 + b_7$
 $x_8 = w_{56} \cdot x_5 + w_{68} \cdot x_6 + b_8$
 return x_7, x_8

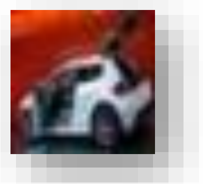
Postcondition

$$x_7 > x_8$$

Either prove that the network output satisfies the postcondition for all inputs in the pre-condition or find a counterexample

Robustness against adversarial perturbations

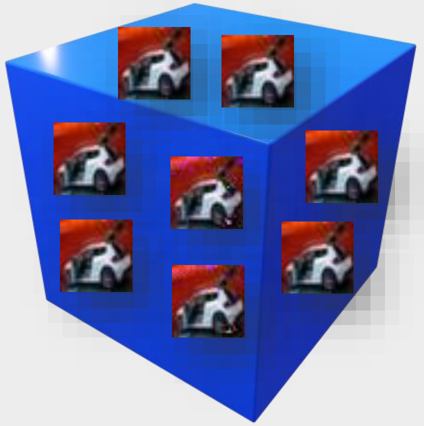
I_0



Network correctly classifies I_0 as "car"

L_∞-ball around
I_0 of radius ϵ

$0.6 \leq x_0 \leq 0.65$
$0.55 \leq x_1 \leq 0.6$
....



ϕ with $\epsilon = 8/255$

ψ: network classifies image as "car"

Verification of Neural Networks

Incomplete	Abstract interpretation: Box, Zonotope, DeepPoly
Complete	Mixed Integer Linear Programming (MILP) SMT solvers (Reluplex)

Active area of research with annual competition: VNNComp
Current winner: alpha-beta crown

Reluplex: An SMT based approach

Katz, Guy, et al. "Reluplex: An efficient SMT solver for verifying deep neural networks." International Conference on Computer Aided Verification (CAV), 2017.

The Constraint Satisfaction Problem

Set of variables V
Atomic predicate:

- Linear inequality of the form $p = (\sum_{v_i \in V} a_i v_i \leq c_i)$
- ReLU equation of the form $p = (v_i = ReLU(v_j))$

Given a conjunction of atomic predicates $\varphi = p_1 \wedge \cdots \wedge p_t$ decide if φ is satisfiable

- φ_N : conjunction of atomic predicates gives relation between input, output, and hidden variables of N.
- Pre-condition is given by a conjunction of linear inequalities $I = \{x \mid x \vDash \varphi_I\}$ and post-condition is a disjunction of linear (strict) inequalities $F = \{z \mid z \vDash \varphi_F\}$
- Sufficient to check if $\varphi_N \wedge \varphi_I \wedge \neg\varphi_F$ is satisfiable!

The Simplex Algorithm

- Solves satisfiability of conjunction of linear inequalities

$$\varphi = v_1 + v_2 \leq -5 \land v_1 - v_2 \geq 3$$

- Step 1: Construct an initial **configuration**

Add one new variable for each inequality

$$\mathrm{V} = \{v_1, v_2, v_3, v_4\}$$

Set of **basic** variable $B \subseteq V$
Initially it is all new variables

$u: V \to \mathbb{R} \cup \{\infty\}$
$l: V \to \mathbb{R} \cup \{-\infty\}$
Upper and lower bounds on variable

- $\boldsymbol{B} = \{v_3, v_4\}$
- $\boldsymbol{T}$: $v_3 = v_1 + v_2$ and $v_4 = v_1 - v_2$
- $\boldsymbol{u}(v_3) = -5$ and $\boldsymbol{l}(v_4) = 3$
- $\boldsymbol{\alpha}(v_i) = 0$ forall $v_i \in V$

Tableau $\boldsymbol{T}$ contains one equation per basic variable**; RHS only has non-basic variables**

Valuation satisfies T

Goal of the algorithm: Update current values of non-basic variables to meet all lower/upper bounds

Derivations

Simplex provides rules of the following kind for modifying a configuration $C = (V, B, T, u, l, \alpha)$

$$\frac{\text{Some condition on } C}{\text{SAT}} \qquad \frac{\text{Some condition on } C}{\text{UNSAT}} \qquad \frac{\text{Some condition on } C}{\text{Modified configuration } C'}$$

Apply rules until a SAT or UNSAT is derived

$$C_0 \xrightarrow{R_1} C_1 \xrightarrow{R_2} \cdots \xrightarrow{R_h} C_h \longrightarrow \text{SAT or UNSAT}$$

Slack Variables

Coefficient of v_j in RHS defining v_i

$$\mathtt{slack}^+(v_i) = \{v_j \notin B \mid (T_{i,j} > 0 \wedge \alpha(v_j) < u(v_j)) \vee (T_{i,j} < 0 \wedge \alpha(v_j) > l(v_j))\}$$

$$\mathtt{slack}^-(v_i) = \{v_j \notin B \mid (T_{i,j} < 0 \wedge \alpha(v_j) < u(v_j)) \vee (T_{i,j} > 0 \wedge \alpha(v_j) > l(v_j))\}$$

$slack^+(v_i)$: Variables on RHS defining v_i whose values can be changed to **increase** the value of v_i
e.g. variable v_j has positive coefficient and its current value is less than its upper bound

$slack^-(v_i)$: Variables on RHS defining v_i whose values can be changed to **decrease** the value of v_i

Slack Variables

$$\texttt{slack}^+(v_i) = \{v_j \notin B \mid (T_{i,j} > 0 \wedge \alpha(v_j) < u(v_j)) \vee (T_{i,j} < 0 \wedge \alpha(v_j) > l(v_j))\}$$

$$\texttt{slack}^-(v_i) = \{v_j \notin B \mid (T_{i,j} < 0 \wedge \alpha(v_j) < u(v_j)) \vee (T_{i,j} > 0 \wedge \alpha(v_j) > l(v_j))\}$$

$\mathrm{V} = \{v_1, v_2, v_3, v_4\}$

- $\boldsymbol{B} = \{v_3, v_4\}$
- $\boldsymbol{T}$: $v_3 = v_1 + v_2$ and $v_4 = v_1 - v_2$
- $\boldsymbol{u}(v_3) = -5$ and $\boldsymbol{l}(v_4) = 3$
- $\boldsymbol{\alpha}(v_i) = 0$ forall $v_i \in V$

$slack^+(v_3) = \{v_1, v_2\}$

If we have $u(v_1) = 0$ then
$slack^+(v_3) = \{v_2\}$

Simplex Rule: Successful Termination

$$\frac{\forall\, v_i \in V, \qquad l(v_i) \leq \alpha(v_i) \leq u(v_i)}{\text{SAT}} \quad \texttt{Success}$$

Current valuation meets all lower/upper bounds: satisfying assignment found

Simplex Rule: Unsuccessful Termination

$$\frac{v_i \in B, \qquad (\alpha(v_i) < l(v_i) \wedge \texttt{slack}^+(v_i) = \emptyset) \vee (\alpha(v_i) > u(v_i) \wedge \texttt{slack}^-(v_i) = \emptyset)}{\texttt{UNSAT}} \quad \texttt{Failure}$$

There is a basic variable for which current value must be increased/decreased to meet lower/upper bound constraint but no such update of RHS vars is possible

Simplex Rule: Pivot

$$\frac{v_i \in B, \qquad \alpha(v_i) < l(v_i), \qquad v_e \in \texttt{slack}^+(v_i)}{T := \texttt{pivot}(T, v_i, v_e), \qquad B := B \cup \{v_e\} \setminus \{v_i\}} \quad \texttt{Pivot}_1$$

$$\frac{v_i \in B, \qquad \alpha(v_i) > u(v_i), \qquad v_e \in \texttt{slack}^-(v_i)}{T := \texttt{pivot}(T, v_i, v_e), \qquad B := B \cup \{v_e\} \setminus \{v_i\}} \quad \texttt{Pivot}_2$$

If: A basic variable's value needs to be increased/decreased to meet lower/upper bound
and there is a possible variable on RHS whose value be changed for this purpose
Then make it non-basic by swapping their roles using pivot

Pivot Example

Pivot: Allows replacing basic variable with a non-basic variable

$$T = \boxed{\begin{array}{l} v_3 = v_1 + v_2 \\ v_4 = v_1 - v_2 \end{array}} \xrightarrow{\text{pivot}(T, 3,1)} \boxed{\begin{array}{c} v_1 = v_3 - v_2 \\ v_4 = (v_3 - v_2) - v_2 \end{array}} = T'$$

Simplex Rule: Update

$$\frac{v_j \notin B, \qquad \alpha(v_j) < l(v_j) \vee \alpha(v_j) > u(v_j), \qquad l(v_j) \leq \alpha(v_j) + \delta \leq u(v_j)}{\alpha := \texttt{update}(\alpha, v_j, \delta)} \quad \texttt{Update}$$

For a non-basic variable, if its current value is less/greater than lower/upper bound
then increase/decraese it to meet the bound

Update Example

Update: Allows updating value of a non-basic variable

α: $v_1 = v_2 = v_3 = v_4 = 0$

$\xrightarrow{\text{update}(v_3, -5)}$

$v_1 = v_3 = v_4 = -5$
$v_2 = 0$: α'

$v_1 = v_3 - v_2$
$v_4 = v_3 - 2v_2$

$T = T'$

Simplex Algorithm in SMT Solver

- Starting with initial configuration, keep applying pivot/update rules until the termination condition holds

- Multiple rules may be applicable in a given configuration
  - Derivation tree captures all possible branches of rule applications

- Key engineering details of implementation
  - Which rule to choose in a given configuration
  - Choice of rules affects time to termination but does not require backtracking
  - How to apply rules efficiently (e.g. keeping track of slack variables)

Soundness and Completeness

SOUNDNESS: If there is a derivation to SAT (or UNSAT), φ is satisfiable (or not).

COMPLETENESS: There is always a derivation to either SAT or UNSAT.

Constraints in Neural network Verification

Precondition

$\forall x_1, x_2 . l_1 \leq x_1 \leq u_1, l_2 \leq x_2 \leq u_2$

DNN f

def $f(x_1, x_2)$:
  $x_3 = w_{13} \cdot x_1 + w_{23} \cdot x_2 + b_3$
  $x_4 = w_{14} \cdot x_1 + w_{24} \cdot x_2 + b_4$
  $x_5 = \mathrm{ReLU}(x_3) = \max(0, x_3)$
  $x_6 = \mathrm{ReLU}(x_4) = \max(0, x_4)$
  $x_7 = w_{57} \cdot x_5 + w_{67} \cdot x_6 + b_7$
  $x_8 = w_{56} \cdot x_5 + w_{68} \cdot x_6 + b_8$
  return x_7, x_8

Postcondition

$x_7 > x_8$

Simplex to Reluplex

- Solves satisfiability of conjunction of linear inequalities **and ReLU equations**.

$$\varphi = v_1 + v_2 \leq -5 \wedge v_1 - v_2 \geq 3 \wedge v_1 = ReLU(v_2)$$

- Step 1: Construct an initial **configuration** $C = (V, B, T, u, l, \alpha, \boldsymbol{R})$

$$\mathrm{V} = \{v_1, v_2, v_3, v_4\}$$

- $\boldsymbol{B} = \{v_3, v_4\}$
- $\boldsymbol{T}$: $v_3 = v_1 + v_2$ and $v_4 = v_1 - v_2$
- $\boldsymbol{u}(v_3) = -5$ and $\boldsymbol{l}(v_4) = 3$
- $\boldsymbol{\alpha}(v_i) = 0$ forall $v_i \in V$
- $\boldsymbol{R} = \{(v_1 = ReLU(v_2))\}$

Set of **ReLU constraints** R

Modified Successful Termination Test

$$\frac{\forall v_i \in V.\ l(v_i) \leq \alpha(v_i) \leq u(v_i), \qquad \forall (v_f = \mathrm{ReLU}(v_b)) \in R.\ \alpha(v_f) = \mathrm{ReLU}(\alpha(v_b))}{\mathtt{SAT}} \quad \mathtt{ReluSuccess}$$

Current valuation meets all lower/upper bounds and all ReLU constraints

Additional Pivot Rule

$$\frac{v_i \in B, \qquad \exists v_j.\ (v_j = \mathrm{ReLU}(v_i)) \in R \vee (v_i = \mathrm{ReLU}(v_j)) \in R, \qquad v_e \notin B, T_{i,e} \neq 0}{T := \mathtt{pivot}(T, v_i, v_e), \qquad B := B \cup \{v_e\} \setminus \{v_i\}} \ \mathtt{ReluPivot}$$

If a basic variable v_i is involved in a ReLU constraint,
Then swap it's a role with a non-basic variable v_e in its RHS with non-zero coefficient using pivot

Additional Update Rules

$$\frac{v_j \notin B, \qquad (v_j = \text{ReLU}(v_i)) \in R, \qquad \alpha(v_j) \neq \text{ReLU}(\alpha(v_i))}{\alpha := \texttt{update}(\alpha, v_j, \text{ReLU}(\alpha(v_i)) - \alpha(v_j))} \; \texttt{Update}_f$$

$$\frac{v_i \notin B, \qquad (v_j = \text{ReLU}(v_i)) \in R, \qquad \alpha(v_j) \neq \text{ReLU}(\alpha(v_i)), \qquad \alpha(v_j) \geq 0}{\alpha := \texttt{update}(\alpha, v_i, \alpha(v_j) - \alpha(v_i))} \; \texttt{Update}_b$$

If a non-basic variable v_i is involved in a ReLU constraint that's violated by its current value,
Then update its value to satisfy the ReLU constraint

New Split Rule

$$\frac{(v_j = \mathrm{ReLU}(v_i)) \in R, \qquad l(v_i) < 0, \qquad u(v_i) > 0}{u(v_i) := 0 \qquad l(v_i) := 0} \quad \texttt{ReluSplit}$$

For the constraint v_j = ReLU(v_i) = max (0, v_i), when v_i can be both positive and negative we have two cases:

Case 1: v_i is positive (achieved by setting its lower bound to 0)

Case 2: v_i is negative (achieved by setting its upper bound to 0)

The two cases create two branches in the derivation tree

A priori we don't know which one will lead to success (so may require backtracking in proof search)

Case split is due to non-linearity of ReLU and crux of computational difficulty

ReluPLEX Algorithm in SMT Solver

- Starting with initial configuration, keep applying rules until SAT leaf found or all branches caused by split lead to UNSAT

- Multiple rules may be applicable in a given configuration
  - Derivation tree captures all possible branches of rule applications
  - Key difference with Simplex: Backtracking (exploring different branches) may be required!
  - Key benefit of Reluplex: Case split is demand driven and happens only when necessary

- Key engineering details of implementation
  - Which rule to choose in a given configuration
  - Choice of rules affects backtracking and time to termination
  - How to apply rules efficiently and how to backtrack efficiently

Soundness and Completeness

SOUNDNESS: If there is a derivation tree with at least one SAT leaf, φ is satisfiable.
If there is a derivation tree with all UNSAT leaves, φ is not satisfiable.

COMPLETENESS: There is always a derivation tree in which every leaf is either SAT or UNSAT.

Comparison with existing SMT solvers

- Can encode $v_1 = ReLU(v_2)$ as $(v_2 \geq 0 \wedge v_1 = v_2) \vee (v_2 \leq 0 \wedge v_1 = 0)$.
- Existing SMT solvers perform many case splits.
- Reluplex can avoid/reduce splitting by using new pivot and update rules first.

	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

Time to termination in seconds with 4 hour timeout

Properties are from case study of neural-network-based controller for collision avoidance protocol ACAS (see ReluPlex paper; Katz et al; CAV 2017)

Experiments

Table Local adversarial robustness tests. All times are in seconds.

	$\delta = 0.1$		$\delta = 0.075$		$\delta = 0.05$		$\delta = 0.025$		$\delta = 0.01$		Total
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Time
Point 1	SAT	135	SAT	239	SAT	24	UNSAT	609	UNSAT	57	1064
Point 2	UNSAT	5880	UNSAT	1167	UNSAT	285	UNSAT	57	UNSAT	5	7394
Point 3	UNSAT	863	UNSAT	436	UNSAT	99	UNSAT	53	UNSAT	1	1452
Point 4	SAT	2	SAT	977	SAT	1168	UNSAT	656	UNSAT	7	2810
Point 5	UNSAT	14560	UNSAT	4344	UNSAT	1331	UNSAT	221	UNSAT	6	20462

SAT : Not Robust
UNSAT : Robust

Neural Network: Fully connected, 8 layers, 300 neurons

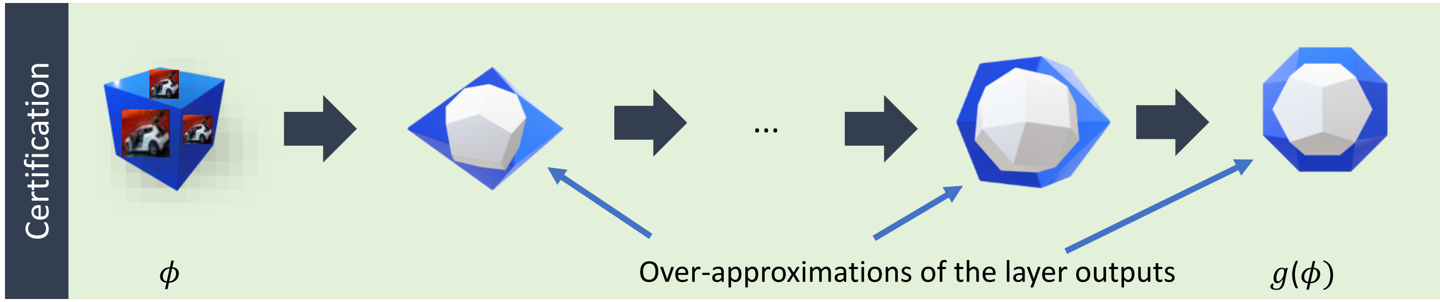
Abstract Interpretation using Boxes

Singh, et al. “Fast and effective robustness certification" NeurIPS, 2018.

Incomplete methods

We will investigate a specific type of incomplete method, based on bound propagation through the neural network. Starting with the initial pre-condition ϕ , we will ``pass'' ϕ through the network, computing a convex over-approximation of the effect of each layer on ϕ. Next, lets look at the “recipe” for certification with bound propagation

Step1: compute convex $g(\phi)$ by propagating ϕ



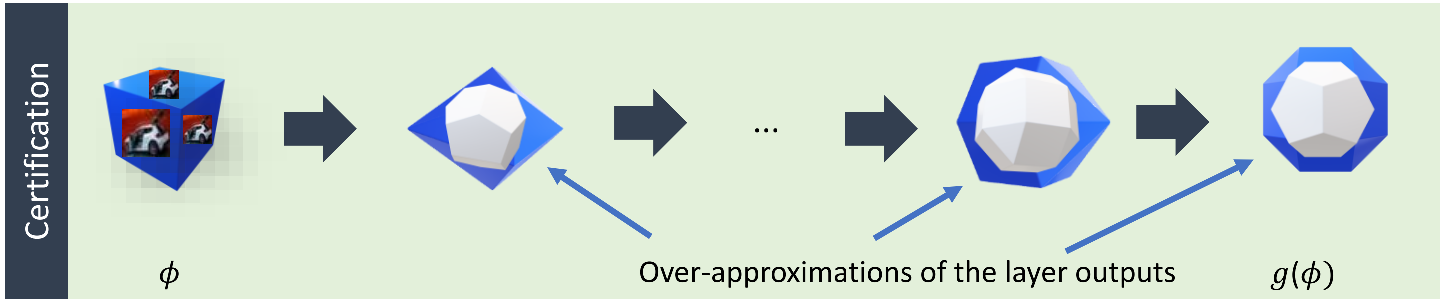
ϕ:
$x_0 = [0.588,0.65]$
$x_1 = [0.545,0.608]$
$x_2 = [0.533,0.596]$
…
$x_{3071} = [0.4,0.463]$

All possible outputs

$g(\phi)$

$o_0 = 0$
$o_1 = 2.60 + 0.015\eta_0 + 0.023\eta_1 + 5.181\eta_2 + \cdots$
$o_2 = 4.63 - 0.005\eta_0 - 0.006\eta_1 + 0.023\eta_2 + \cdots$
…
$o_9 = 0.12 - 0.125\eta_0 + 0.102\eta_1 + 3.012\eta_2 + \cdots$
$\forall i. \eta_i \in [0,1]$

Step2: verify ψ



$g(\phi)$

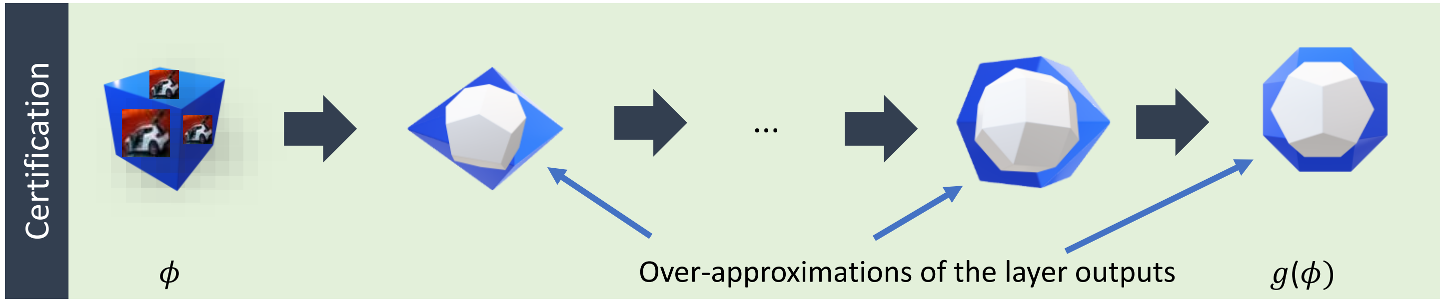
$$
\begin{aligned}
o_0 &= 0\\
o_1 &= 2.60 + 0.015\eta_0 + 0.023\eta_1 + 5.181\eta_2 + \cdots\\
o_2 &= 4.63 - 0.005\eta_0 - 0.006\eta_1 + 0.023\eta_2 + \cdots\\
&\ldots\\
o_9 &= 0.12 - 0.125\eta_0 + 0.102\eta_1 + 3.012\eta_2 + \cdots\\
&\forall i.\, \eta_i \in [0,1]
\end{aligned}
$$

$\boldsymbol{\psi}$: every point in ϕ classifies as car

$\boldsymbol{\psi}$: $\forall l \neq car.\, o_{car} > o_l$

$g(\phi)$ is an over-approximation, hence if we fail to prove ψ , it could be the property is actually violated or there was over-approximation introduced during propagation which prohibits provability.

Key challenge: how to produce convex shapes?

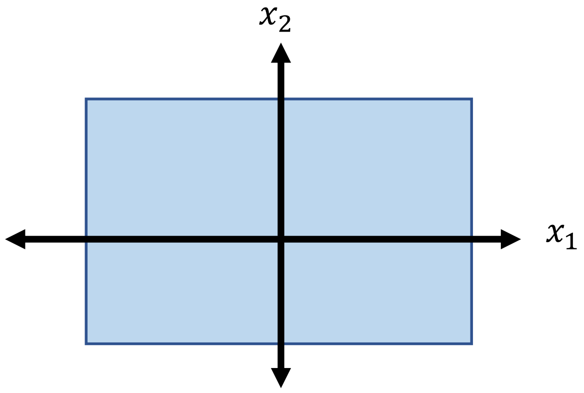


To instantiate incomplete methods which use bound propagation, we need two parts:

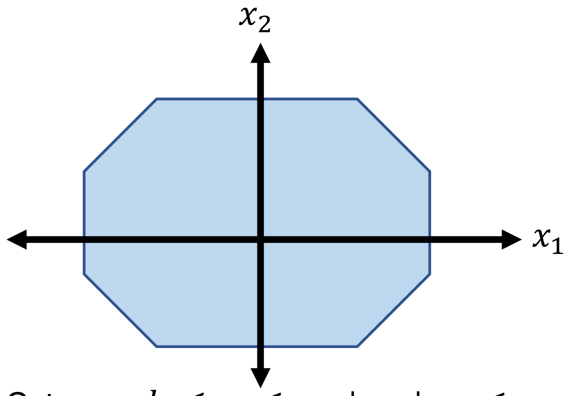
1. What is the convex approximation ? E.g., Box, Zonotope, Polyhedra

2. How are these convex approximations produced? That is, what is the effect of the layer on a given approximation ? This effect is often called an abstract transformer as it transforms abstract shapes.

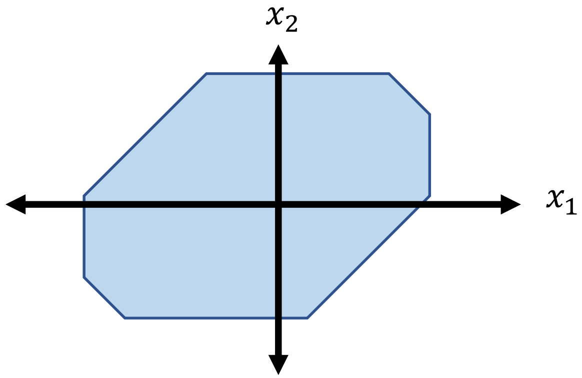
Popular convex shapes



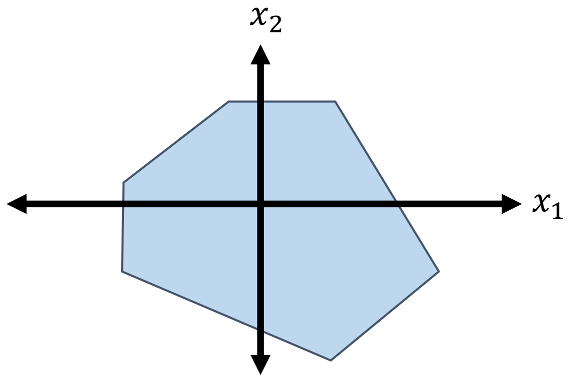
Box: $l_i \leq x_i \leq u_i$



Octagon: $l_i \leq x_i \leq u_i, \pm x_i \pm x_j \leq c_{ij}$



Zonotope: $\hat{x}_i = \alpha_0 + \sum_i \alpha_i \epsilon_i\,, \epsilon_i \in [-1,1]$



Polyhedra: $\sum_i a_i x_i \leq c$

Speed vs. precision tradeoff

A trade-off between approximation and speed exists:

transformers for Box are fast, but imprecise,

while Polyhedra is more precise but has exponential complexity.

Box Abstract Transformer: Addition

$[\boldsymbol{a}, \boldsymbol{b}] \ +^{\#} [\boldsymbol{c}, \boldsymbol{d}] = [\boldsymbol{a} + \boldsymbol{c}, \boldsymbol{b} + \boldsymbol{d}]$ Addition transformer

- Here, $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{R^m}$ where $\forall \boldsymbol{i}. \boldsymbol{a_i} \leq \boldsymbol{b_i}$
- $^{\#}$ denotes the abstract effect of the operation on the box

Example in 1-dim (interval addition) $[1, 3] \ +^{\#} [5, 8] = [6, 11]$

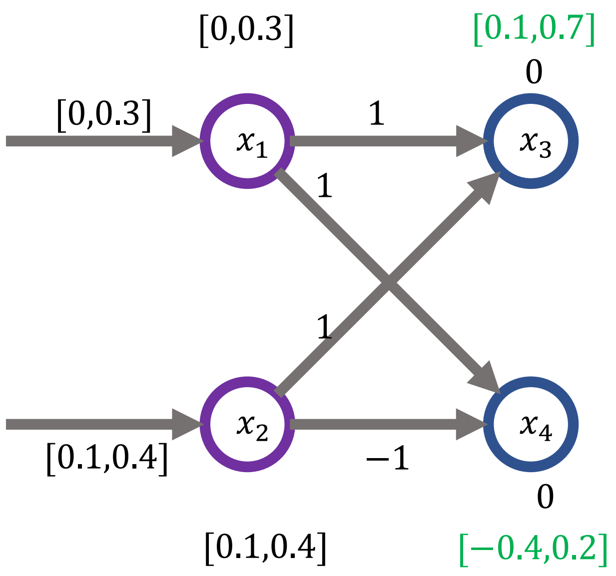
Box Abstract Transformers for ReLU Networks

$[\boldsymbol{a}, \boldsymbol{b}] \; +^{\#} [\boldsymbol{c}, \boldsymbol{d}] = [\boldsymbol{a} + \boldsymbol{c}, \boldsymbol{b} + \boldsymbol{d}]$	Addition transformer
$-^{\#}[\boldsymbol{a}, \boldsymbol{b}] = [-\boldsymbol{b}, -\boldsymbol{a}]$	Negation transformer
$\boldsymbol{ReLU}^{\#}[\boldsymbol{a}, \boldsymbol{b}] = [\boldsymbol{ReLU}(\boldsymbol{a}), \boldsymbol{ReLU}(\boldsymbol{b})]$	ReLU transformer
$\boldsymbol{\lambda}^{\#}[\boldsymbol{a}, \boldsymbol{b}] = [\boldsymbol{\lambda} * \boldsymbol{a}, \boldsymbol{\lambda} * \boldsymbol{b}]$	Multiplication by a constant $\boldsymbol{\lambda} > \boldsymbol{0}$
$[\boldsymbol{a}, \boldsymbol{b}] \; >^{\#} [\boldsymbol{c}, \boldsymbol{d}] \; \boldsymbol{holds \; iff \; a > d}$	Comparison transformer

- Here, $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{R^m}$ where $\forall \boldsymbol{i}.\, \boldsymbol{a_i} \leq \boldsymbol{b_i}$
- $\boldsymbol{ReLU}(\boldsymbol{x}) = \boldsymbol{max}(\boldsymbol{0}, \boldsymbol{x})$
- $^{\#}$ denotes the abstract effect of the operation on the box

Computation with Box transformer

We have 2 pixels (x_1, x_2) as input ranging over $[0, 0.3]$ and $[0.1, 0.4]$



def $f_1(x_1, x_2)$:
  $x_3 = x_1 + x_2$
  $x_4 = x_1 - x_2$
return x_3, x_4

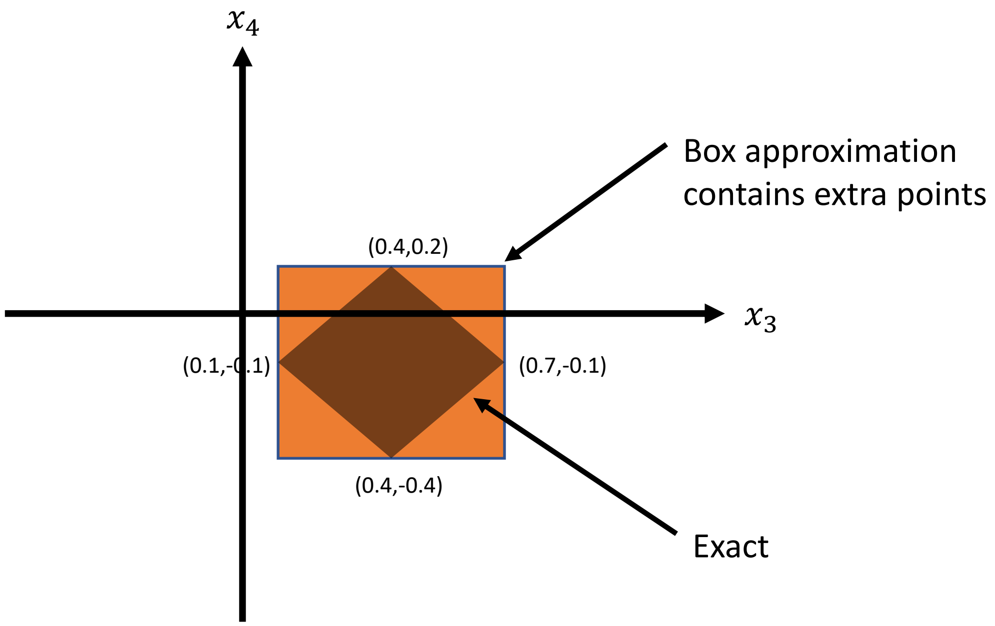
Bounds using Box:

$$0.1 \le x_3 \le 0.7$$
$$-0.4 \le x_4 \le 0.2$$

Exact bounds would be:

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$0 \le x_1 \le 0.3$$
$$0.1 \le x_2 \le 0.4$$

Optimal Box transformer is not exact!

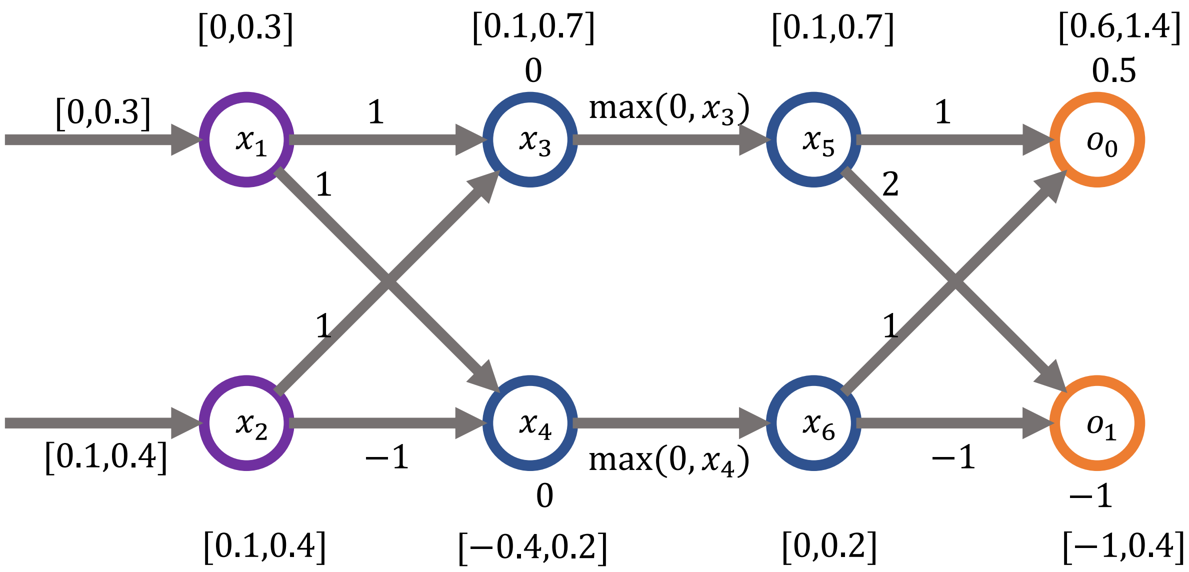


Key Point

Even though the Box abstract transformer for the affine computation is optimal for the Box relaxation, it may not be complete (exact)! Nonetheless, even if not exact, it may be enough to verify the property of interest.

Box succeeds in verifying robustness

We have 2 pixels (x_1, x_2) as input ranging over $[0, 0.3]$ and $[0.1, 0.4]$, we want to prove that $o_0 > o_1$ holds for all inputs



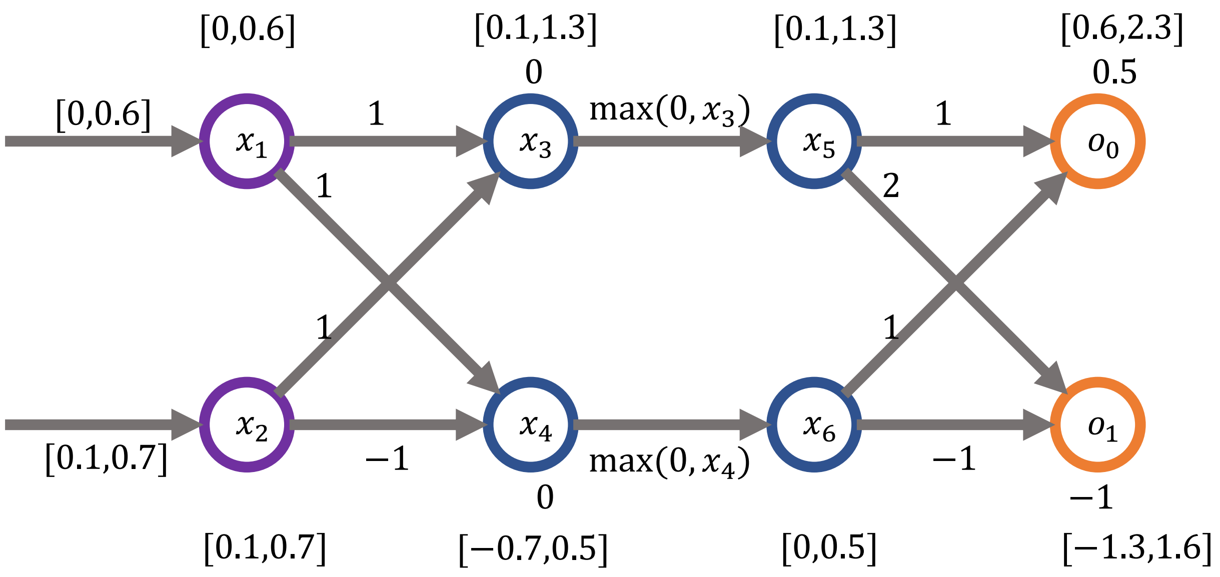
def $f(x_1, x_2)$:

$x_3 = x_1 + x_2$
$x_4 = x_1 - x_2$
$x_5 = \max(0, x_3)$
$x_6 = \max(0, x_4)$
$o_0 = x_5 + x_6 + 0.5$
$o_1 = 2 \cdot x_5 - x_6 - 1$
return o_0, o_1

Using Box, we succeed in proving the network classifies any input in the range as 0.
This is because $[0.6,1.4]$ > $[-1, 0.4]$, provably so.

Box fails in verifying robustness

Let us slightly increase the range of the input pixels to [0, 0.6] and [0.1, 0.7]



def $f(x_1, x_2)$:

$x_3 = x_1 + x_2$

$x_4 = x_1 - x_2$

$x_5 = \max(0, x_3)$

$x_6 = \max(0, x_4)$

$o_0 = x_5 + x_6 + 0.5$

$o_1 = 2 \cdot x_5 - x_6 - 1$

return o_0, o_1

Using Box, we failed to prove the network classifies any input in the range as 0, even though property actually holds. This is because [0.6,2.3] is not > [-1.3, 1.6], provably so.

Robustness Certification

- Instead of Box domain, one can use Zonotopes to get reasonably scalable and more precise approximations
- Constraint solving and abstract domains can be combined in different ways
- Robustness verification can be integrated into training
  - Parameters are updated not only to minimize loss but also to ensure that a verification procedure (e.g. based on Box abstract domain) gives robustness guarantee

Adversarial Robustness Recap

- **Adversarial Examples**
- **Adversarial training**
- **Certified robustness via randomized smoothing**
- **Sample of current research on robustness for LLMs**
- **Formal methods for verified robustness**
  - **Specialized constraint solver ReluPlex for neural network verification**
  - **Verifying robustness by abstract interpretation (box domain)**