Verification of DNNs with Marabou

Omri Isac, The Hebrew University of Jerusalem

Workshop on Logical Methods for Neural Network Analysis (LogicNN), July 2026

Agenda

Learn the basic theory behind Marabou

Run Marabou with simple examples

Understand Maraboupy main objects and try it

Follow along (Linux users)

pip install maraboupy

--- or ---

git clone https://github.com/NeuralNetworkVerification/Marabou/

cd Marabou

mkdir build

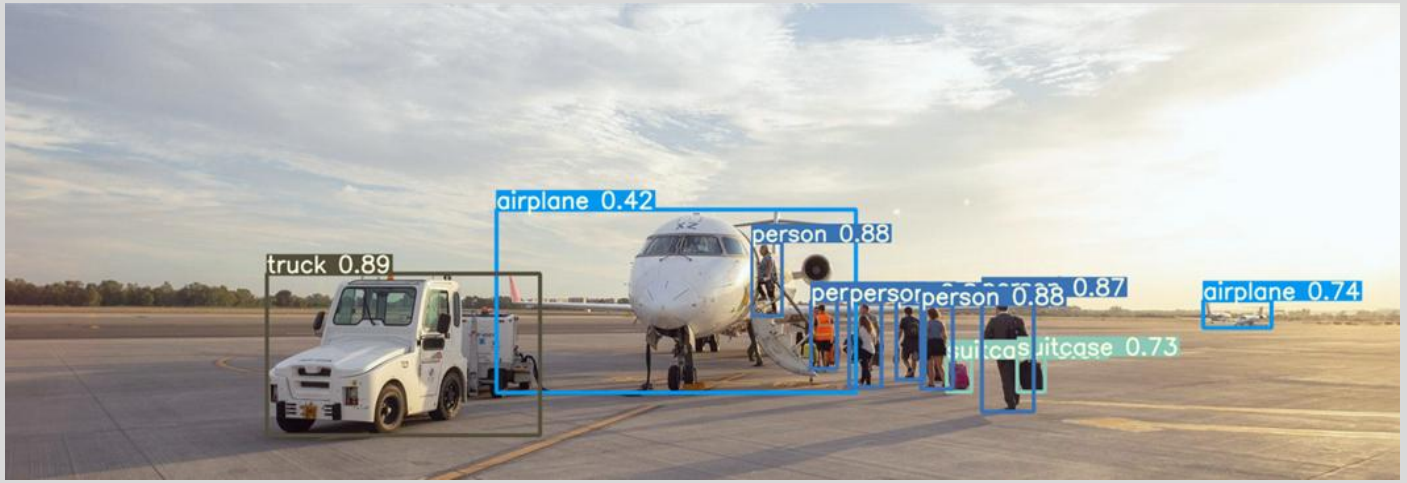
cd build

cmake ..

cmake --build .

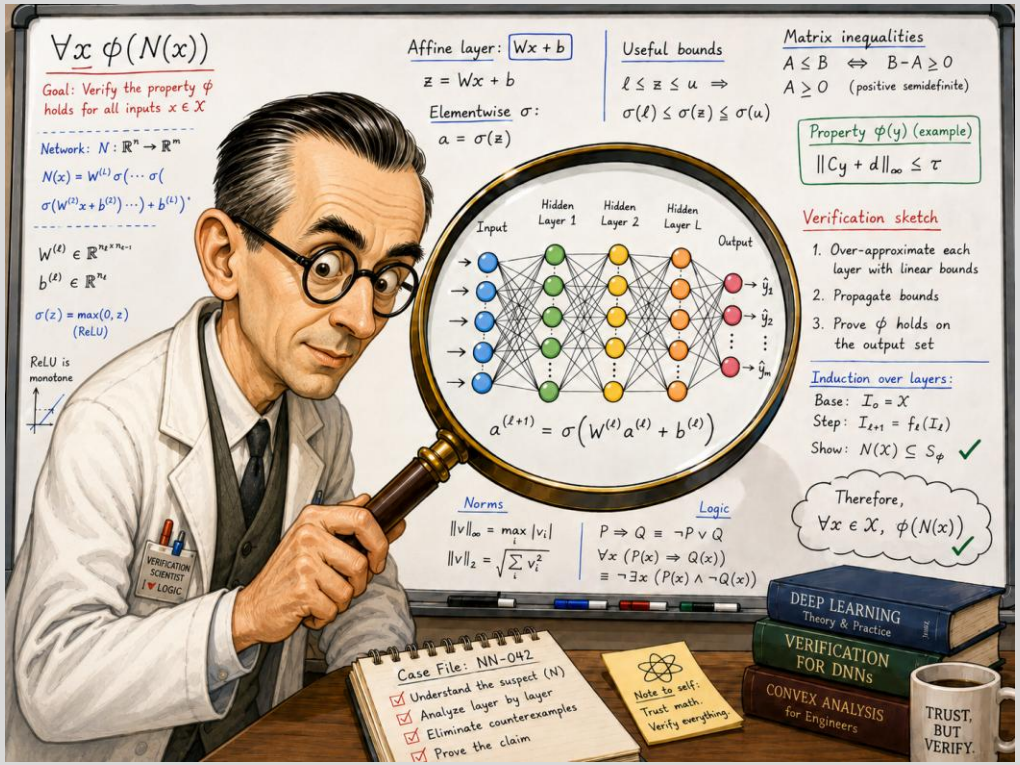
DNN Verification

Deep Neural Networks (DNNs) accomplish groundbreaking results in many fields, including safety-critical.



Unlike traditional software, DNNs are opaque functions, trained over a large set of input and output examples.

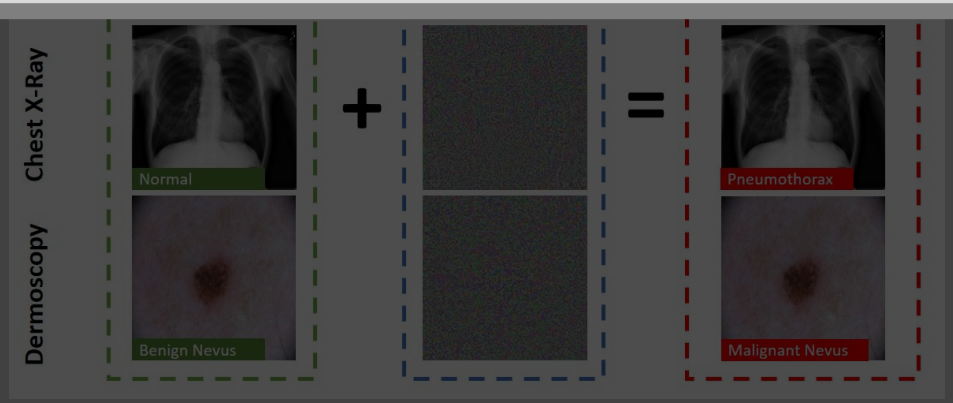
Understanding the behavior of DNNs is then necessary,



and calls for logical analysis for reasoning on edge cases.

This may lead to undesirable behavior of DNNs, which is hard to predict and potentially dangerous.

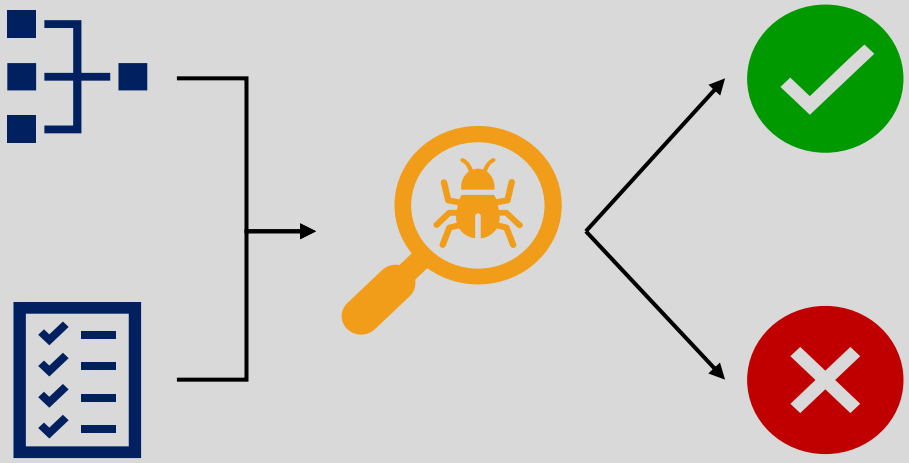
Can we rule out possibility of errors?
Or find hidden errors, if exist



Understanding Adversarial Attacks on Deep Learning Based Medical Image Analysis Systems, Ma et al., 2019

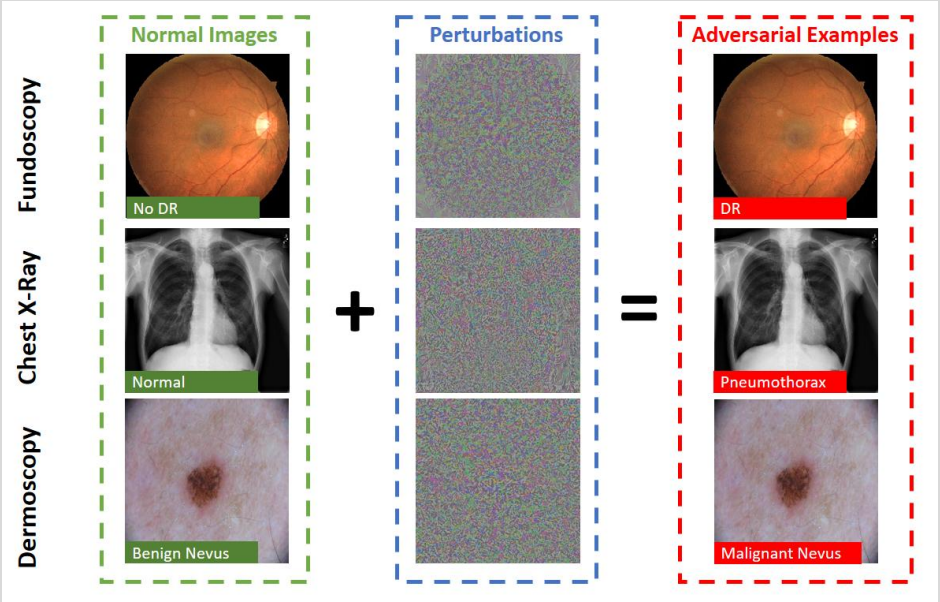
DNN Verification

- Given a DNN $\mathcal{N}$ an input property $\varphi(x)$ and an output property $\psi(y)$, decide whether there exists an input x such that $\varphi(x) \wedge \neg\psi(\mathcal{N}(x))$.
  - If exists, the problem is satisfiable (`SAT`);
  - Otherwise unsatisfiable (`UNSAT`).



Back to Adversarial Examples

- For an image classifier and a given image, we wish to check the network's robustness to small input perturbations.



- Decide whether there exists a small perturbations that alters the image's class.

Local Adversarial Robustness

- Given an image classifier $\mathcal{N}$, an input image α and its classification $c_i \in C$:
- Slight noise perturbation $\varphi(\alpha, \epsilon)$: $\{x_{i,j} = \alpha_{i,j} + e_{i,j} \mid |e_{i,j}| \leq \epsilon\}$
- Classification changes $\neg\psi(c_i)$: $\bigvee_{c_j \neq c_i \in C} c_j \geq c_i$

- Can be generalized in several ways.

Marabou

The Marabou Verifier

- A **mature DNN verifier,** employing multiple state-of-the-art features.

- Operates mainly by casting the **verification problem to SMT-like queries.**

DNN Verification as SMT

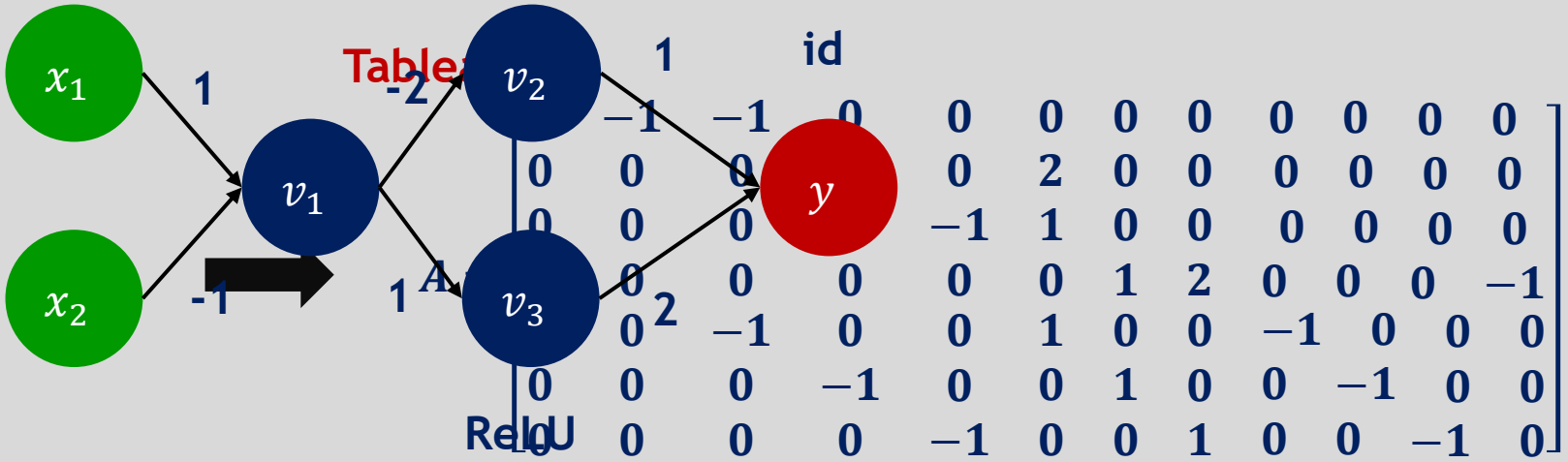
- Decide if there is assignment to V such that:

$$A \cdot V = 0 \text{ for some matrix } A$$

$$l \leq V \leq u \text{ for some bound vectors } l, u$$

Activation constraints over a subset of V

Verification Query Example



Bounds:

$\varphi(x) := x \in [1, 2]^2$

$\neg\psi(y) := y \leq -1$

$u = [2 \; 2 \; 1 \; 0 \; 0 \; 1 \; 0 \; 0 \; 2 \; 1 \; 1 \; -1]$

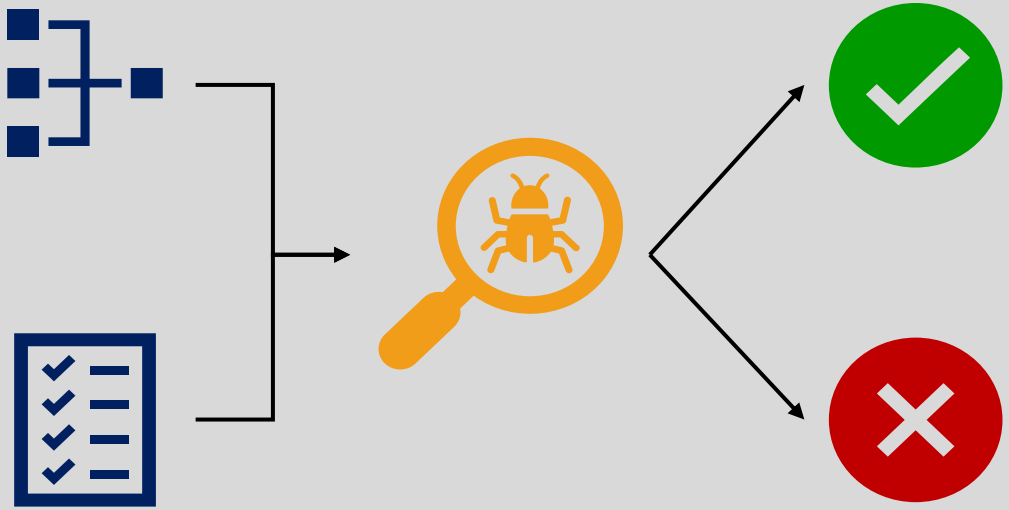
$l = [1 \; 1 \; -1 \; -1 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; -1]$

ReLUs:

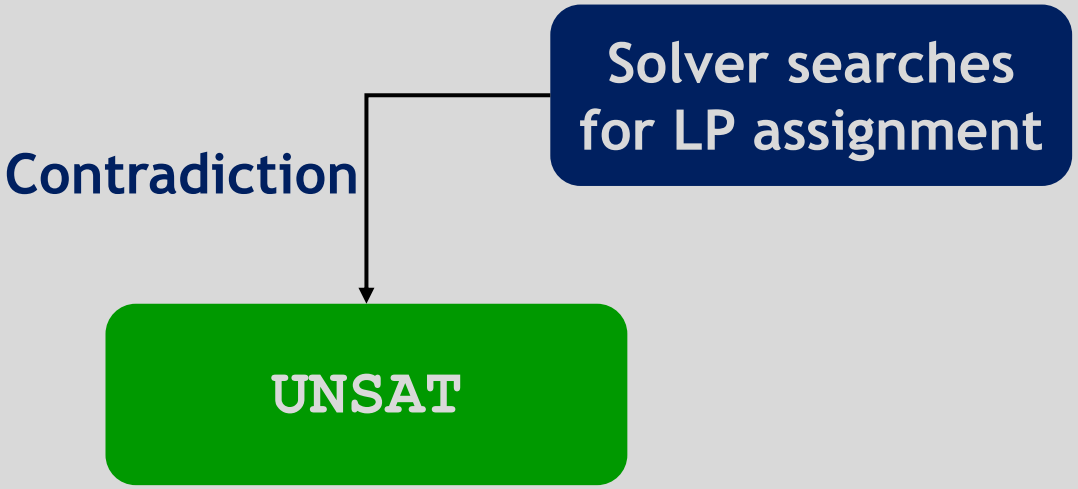
$$f_1 = ReLU(b_1)$$
$$f_2 = ReLU(b_2)$$
$$f_3 = ReLU(b_3)$$

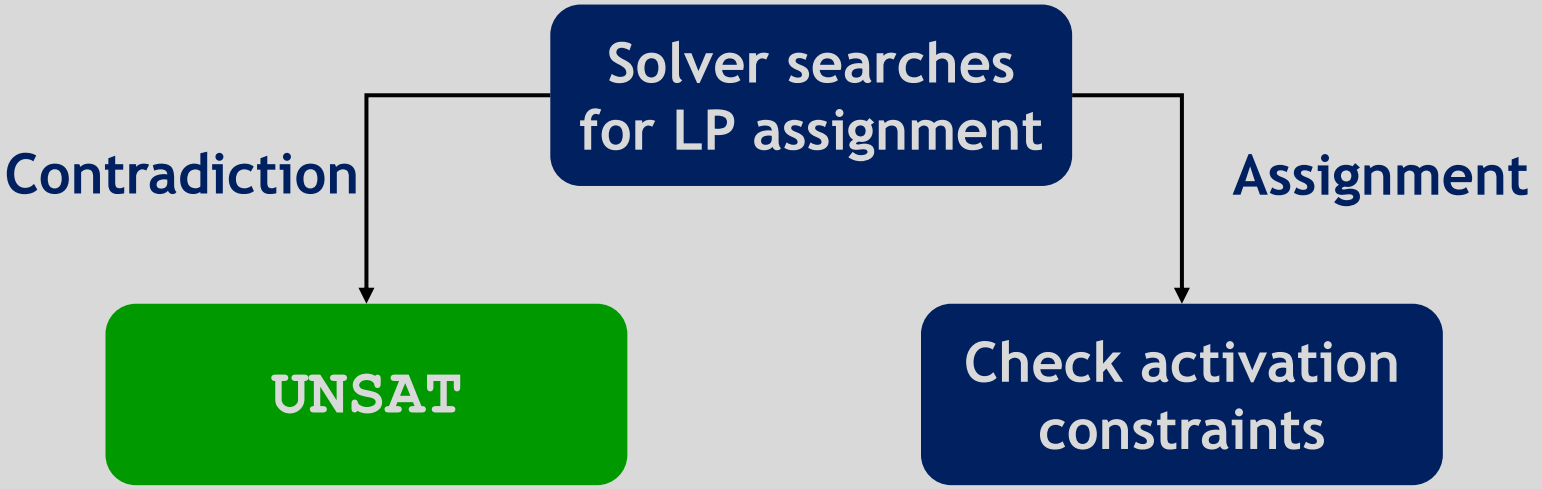
The Basic Approach

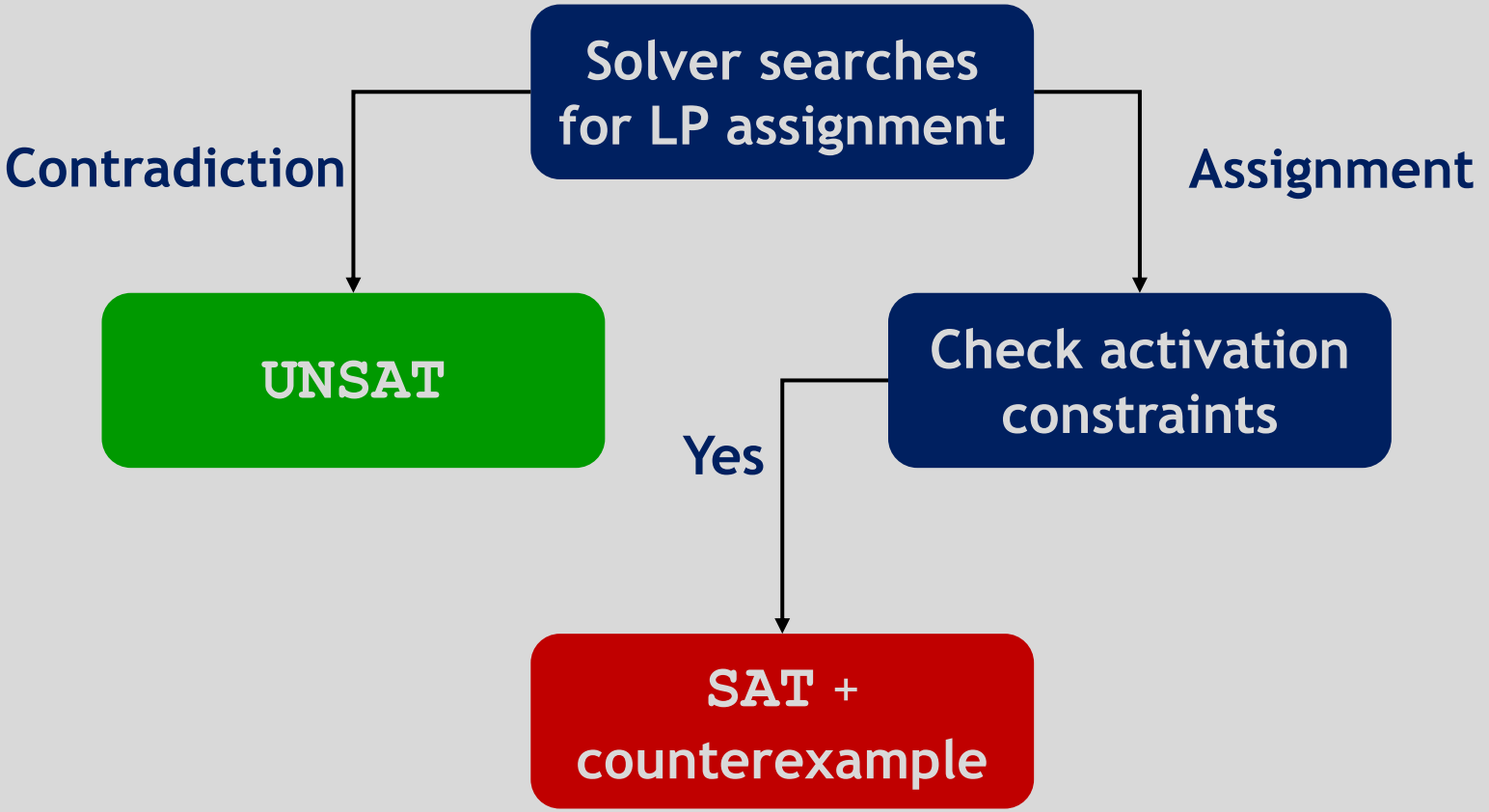
- Marabou searches for a satisfying assignment with LP solvers and splitting approach.

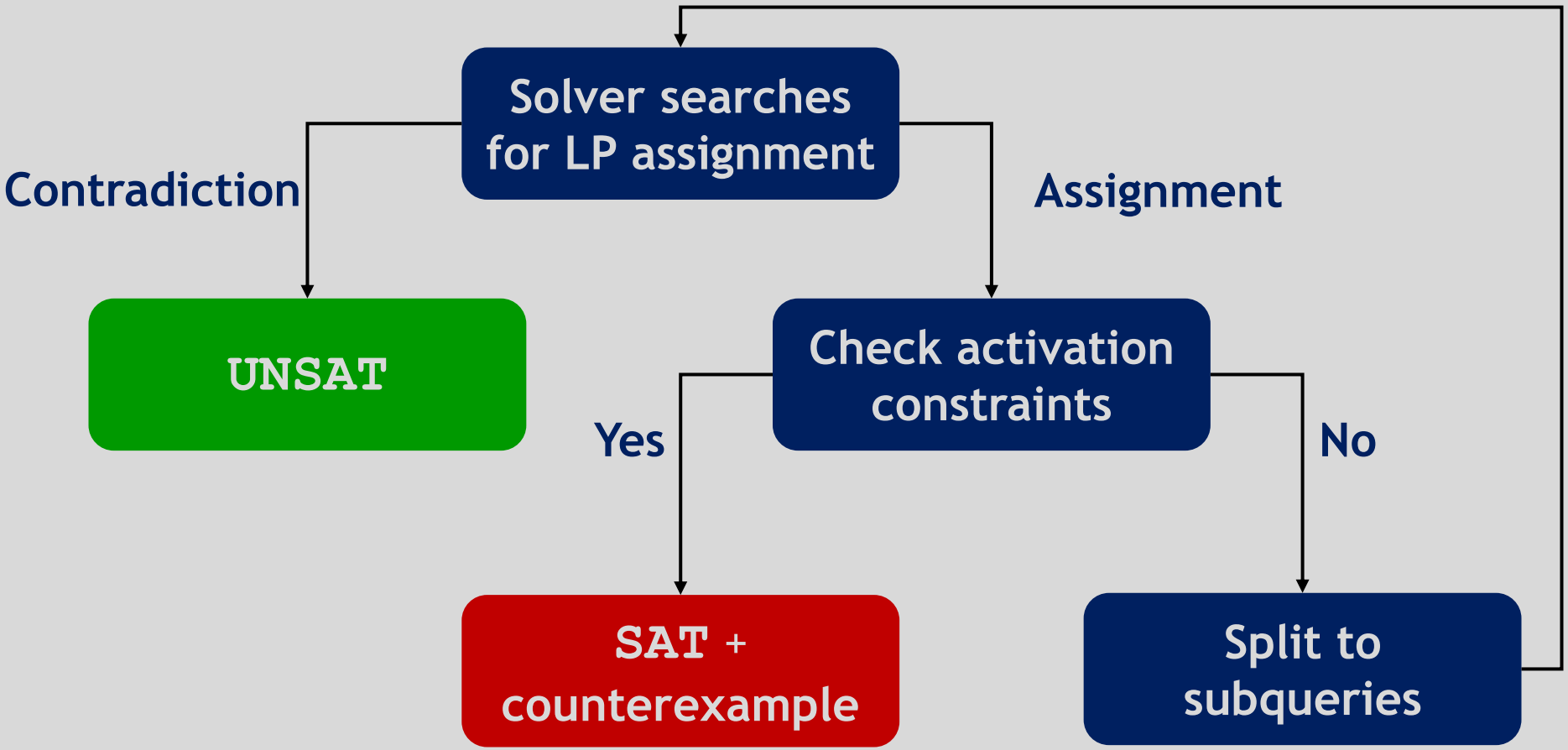


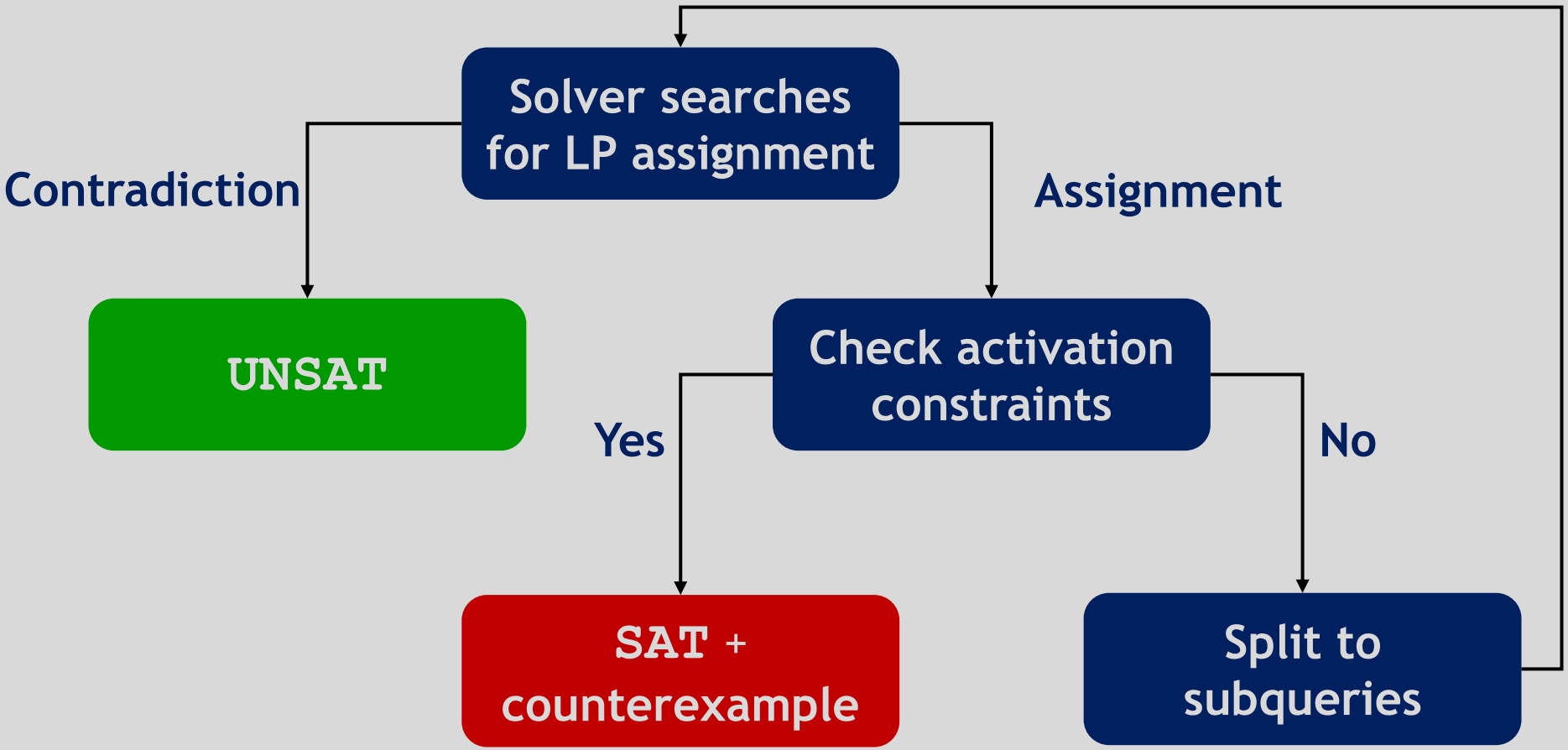
Solver searches for LP assignment


Solver searches
for LP assignment
Contradiction
UNSAT


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check activation constraints


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check activation constraints
Yes
SAT + counterexample


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check activation constraints
Yes
No
SAT + counterexample
Split to subqueries



Splitting creates a tree structure to the search.

The query is UNSAT if and only if all leaves are UNSAT.

The Search Tree

- Potentially, tree size is exponential ~$2^{|\mathcal{N}|}$.
- Possibly unavoidable, as DNN verification is NP-Complete for ReLU-networks [Katz et. al, ’17; Sälzer and Lange, ‘21].
- However, can still be solved in reasonable time, using many optimizations (as in SAT solving).

Pruning The Search Tree

- "Lazy" Splits.
- Effective decision heuristics.
- Overapproximation of bounds, compromising completeness for scalability.

Bound Propagation

- Overapproximates bounds of neurons, as a linear function of neurons in preceding layers.

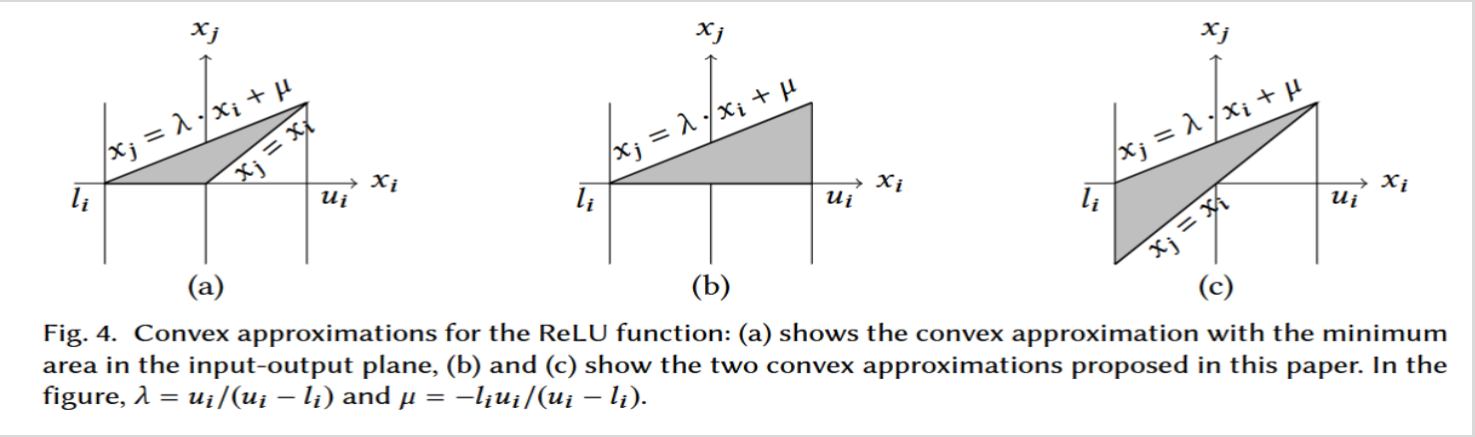
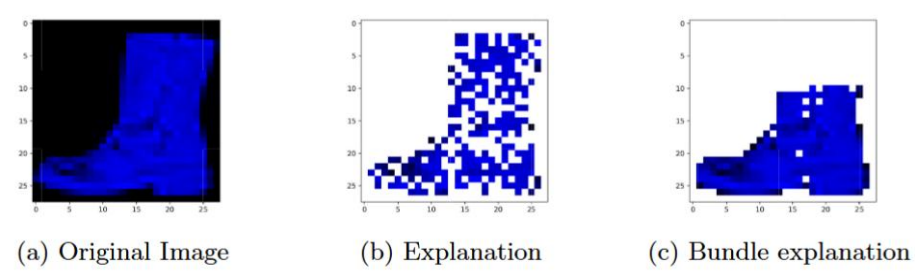


Fig. 4. Convex approximations for the ReLU function: (a) shows the convex approximation with the minimum area in the input-output plane, (b) and (c) show the two convex approximations proposed in this paper. In the figure, $\lambda = u_i/(u_i - l_i)$ and $\mu = -l_i u_i/(u_i - l_i)$.

An Abstract Domain for Certifying Neural Networks, Singh et al., 2019

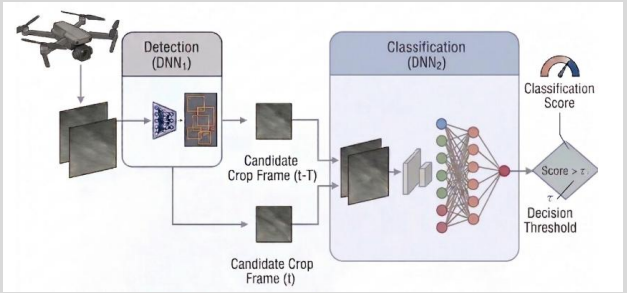
- Eliminates neuron phases (PL) + detects UNSAT early.
- Introduce incompleteness.

Formal Explainability



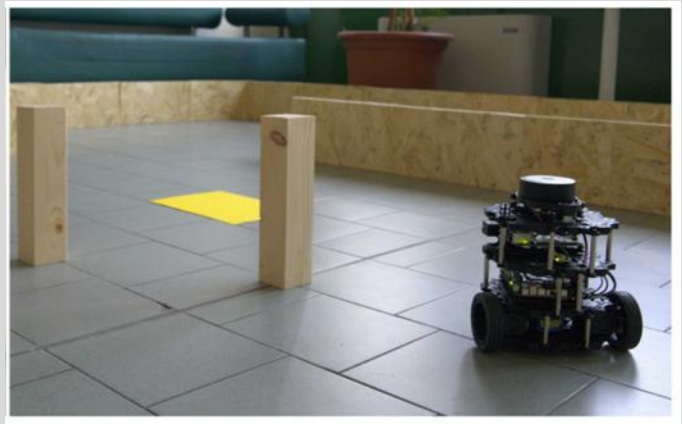
Towards Formal XAI: Formally Approximate Minimal Explanations of Neural Networks, Bassan et al., 2023

Wildfire Detection System



VeriFIRE: an Industrial Case Study in Verifying Consistency Properties for a DNN-Based, Wildfire Detection System, Refaeli et al., SAIV 2026.

Neural Robotics Controllers



Verifying Learning-Based Robotic Navigation Systems, Amir et al., 2023

Network Congestion Control

Verifying Generalization in Deep Learning, Amir et al., 2023

Verifying Learning-Based Robotic Navigation Systems, Amir, Corsi et al., 2023

Using Marabou

The VNNLIB format

A format for defining input-output properties in DNN verification

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)

(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(declare-const Y_3 Real)
(declare-const Y_4 Real)

; Unscaled Input 0: (55947.691, 60760)
(assert (<= X_0 0.679857769))
(assert (>= X_0 0.6))

; Unscaled Input 1: (-3.141592653589793, 3.141592653589793)
(assert (<= X_1 0.5))
(assert (>= X_1 -0.5))

; Unscaled Input 2: (-3.141592653589793, 3.141592653589793)
(assert (<= X_2 0.5))
(assert (>= X_2 -0.5))

; Unscaled Input 3: (1145, 1200)
(assert (<= X_3 0.5))
(assert (>= X_3 0.45))

; Unscaled Input 4: (0, 60)
(assert (<= X_4 -0.45))
(assert (>= X_4 -0.5))

; Unsafe if COC >= 1500. Output scaling is 373.94992 with a bias of 7.518884: (1500 - 7.518884) / 373.94992 = 3.991125
(assert (>= Y_0 3.991125645861615))
```

Undergoing a major update to VNNLIB 2.0 (Roy et al., CAV26)

Demo (onnx + vnnlib)

./Marabou network/path/dnn.onnx property/path/prop.vnnlib

Exploring Options

- Split and Conquer (SnC)
  `--snc --num-workers n`

- Branching heuristics
  `--branch relu-violation/polarity/babsr` and more

- Proofs (if time permits, no SnC)
  `--prove-unsat`

Demo (flags)

./Marabou network/path/dnn.onnx property/path/prop.vnnlib
--options

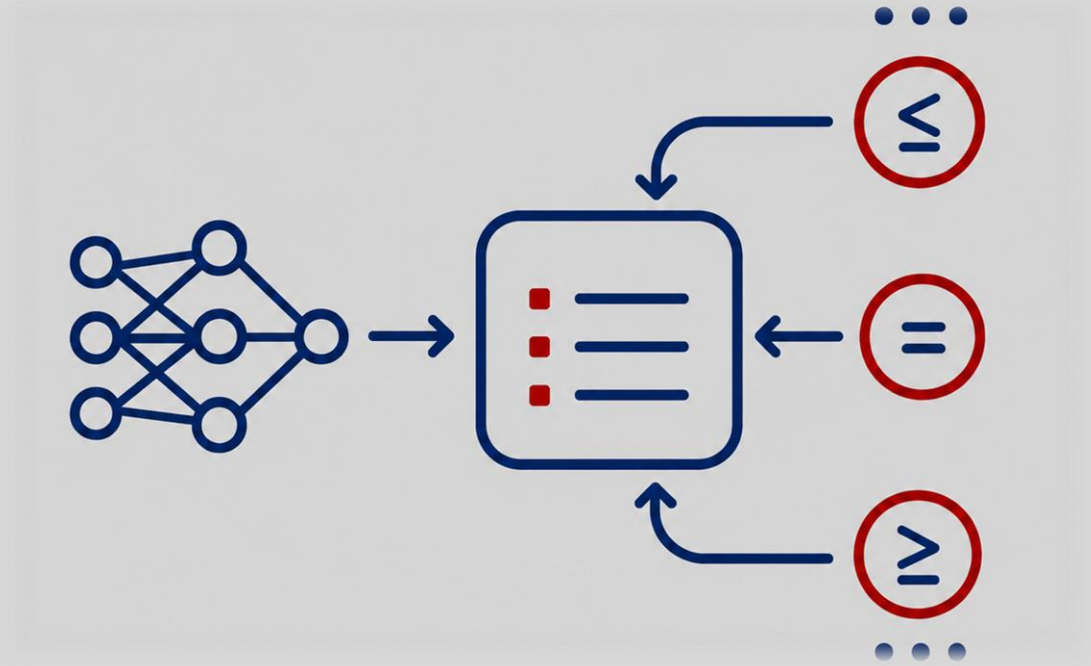
Maraboupy

- Python interface for Marabou, (outdated) documentation:
  - https://neuralnetworkverification.github.io/Marabou/

- Begin with `pip --install maraboupy`

Query Object

- Base object that aggregates constraints:
  - Read a DNN using Marabou.read_onnx(path)
  - Start anew using MarabouCore.InputQuery()



Variables

- Either use the network's inputs and outputs:
  - network.inputVars or outputVars
- Or manually define them:
  - query.setNumberOfVariables(n)
- Bound variables:
  - setUpperBound(index,value), setLowerBound(index,value)

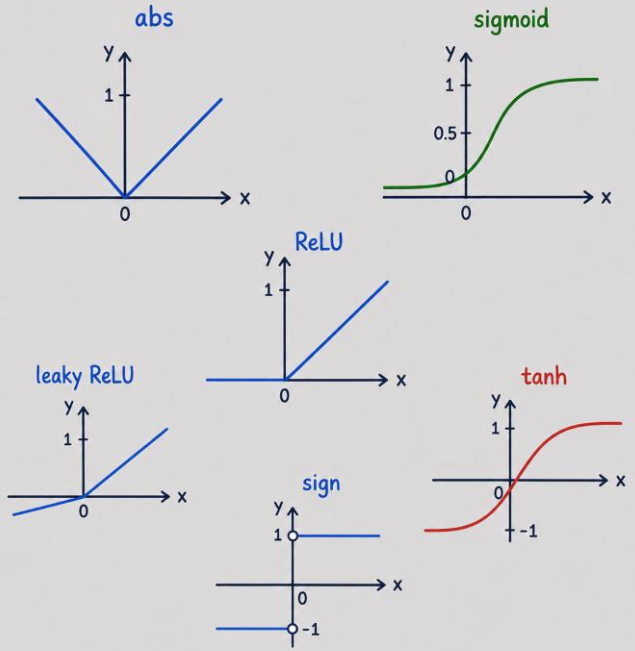
Linear Equations and Inequalities

- Either use the pythonic interface:
  - query.addConstraint(expr)

- Or manually define them:
  - eq = MarabouCore.Equation()
  - eq.addAddend(index, coefficient)
  - eq.scalar = c
  - query.addEquation(eq)

Activation function

- Dedicated function for each piecewise-linear constraint:
  - MarabouCore.addReluConstraint(query, input, output)
  - MarabouCore.addSignConstraint(query, input, output)
  - …



Solving the query

- Define solving options (verbosity, heuristics, etc.):
  - Marabou.createOptions(options)

- Call the solving function and analyze its results:
  - Marabou.solve(query)

https://github.com/OmriIsacHUJI/MarabouTutorial

Your Turn