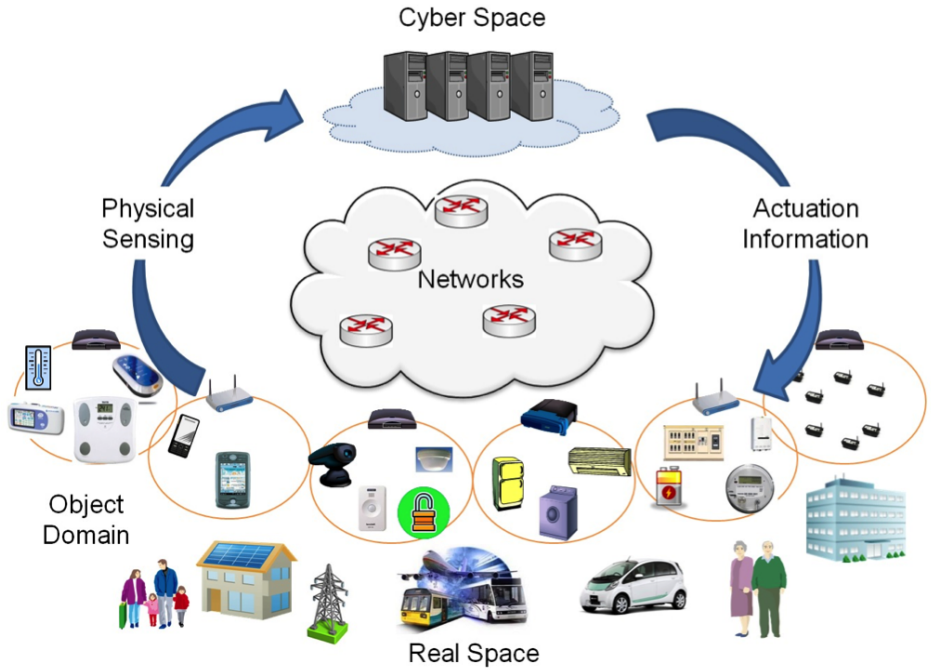
INF641. Introduction to the Verification of Neural Networks

Lecture 1. Introduction and abstraction-based verification

Eric Goubault and Sylvie Putot

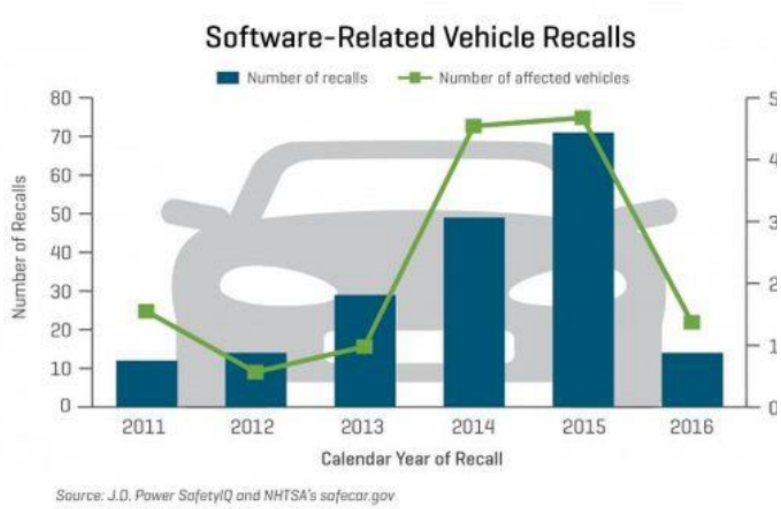
Context: Reliable Artificial Intelligence in Cyber-Physical Systems



- Cyber-Physical Systems? Collaborating devices that interact with their environment, with sensing, computation, communication and control capabilities
  - Examples: autonomous driving, autonomous manufacturing and delivery, medical devices (e.g. pacemakers, insulin pumps), energy production (e.g. smart grids)
- Artificial Intelligence? Learning-enabled algorithms are increasingly used in such systems (e.g. for perception, control)
- Reliable AI? Most of these systems are safety-critical, safety must be assessed

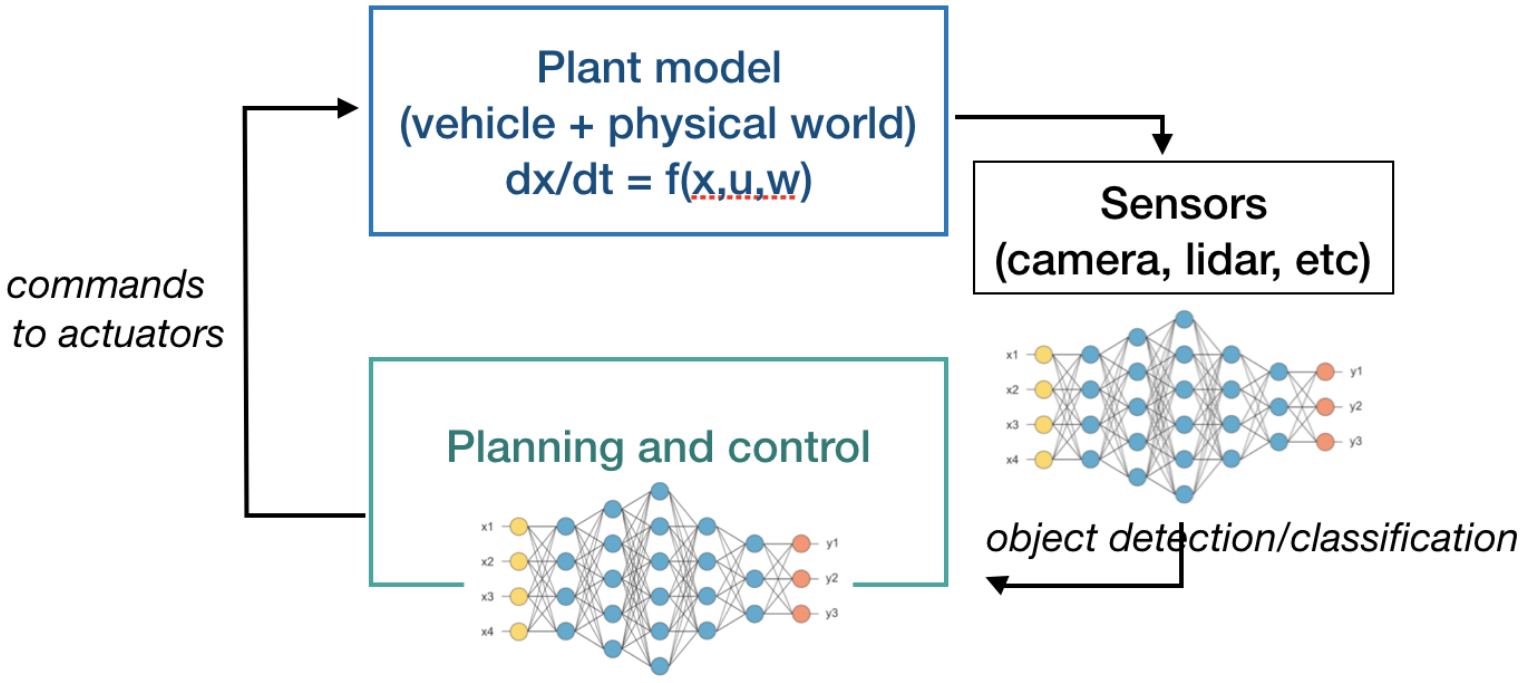
There is a need for safe design !

Fully autonomous cars "soon" ... but their safety remains a challenge!



Warning: Traffic-Aware Cruise Control can not detect all objects and may not brake/ decelerate for stationary vehicles, especially in situations when you are driving over 50 mph (80 km/h) and a vehicle you are following moves out of your driving path and a stationary vehicle or object is in front of you instead. Always pay attention to the road ahead and stay prepared to take immediate corrective action. Depending on Traffic-Aware Cruise Control to avoid a collision can result in serious injury or death. In addition, Traffic-Aware Cruise Control may react to vehicles or objects that either do not exist or are not in the lane of travel, causing Model X to slow down unnecessarily or inappropriately.

Trusting Autonomous CPS

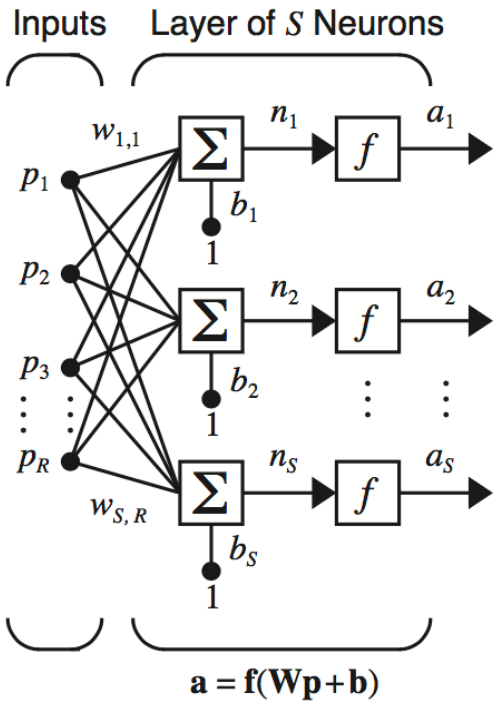


Reliable AI in autonomous CPS ?

- Perception: object detection and classification
  - should be robust to change in lighting, physical attacks, adversarial noise
- Planning and control
  - Robots need to operate in unknown, uncertain and dynamic environments
  - Trained using reinforcement learning, data-driven controller synthesis, etc
- How do we specify and analyze a neural network ?

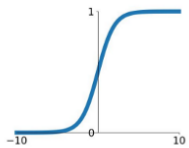
Feedforward neural network: definition

- Succession of layers (inner ones are “hidden”) consisting of simple neurons
- Each layer = a linear transform followed by a non linear activation function
- Wide variety of neural networks : convolutional, feedforward, recurrent, etc. with a wide variety of “activation” functions and applications



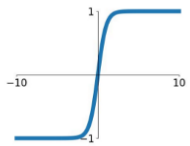
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



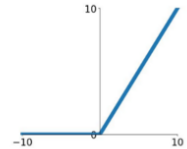
tanh

$\tanh(x)$



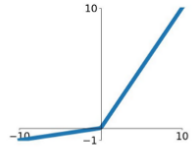
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

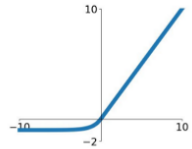


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Universal approximation guarantee

Can approximate a continuous function on a compact set to arbitrary accuracy

Neural networks in autonomous systems

Focus on applications for robotics and autonomous systems/vehicles

- perception: object detection and classification (obstacle, lane marking, etc)
- planning and control: path planning, decision making, model identification or controller (e.g. Airplane flight management system, with uncertainties from wind, component failures, obstacles, other aiplanes...)
- end-to-end learning: a unique network for the full system, from sensors to actuators

Neural network control approaches

- Neural network as function approximator: predict the response of a nonlinear plant over a time horizon (system/model identification)
- Neural network as controller (e.g. trained to approximate the behavior of a traditional controller)

Specifying the expected behavior of neural networks

Many applications lack specifications

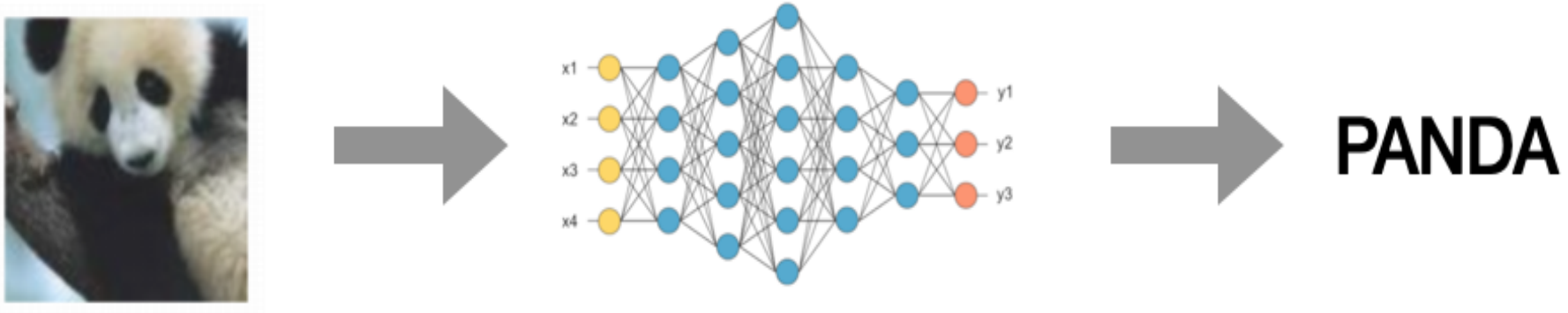
- For example, if the specification is that the network must recognize a stop sign if any, how do we mathematically specify this?

Some useful analyses

- Robustness to disturbances
- Available specification of safety properties: input-output relationships
- Properties on the full control loop behavior: finite-horizon properties (reach-avoid properties under uncertainties), infinite-horizon properties (stability, viability kernels, etc)

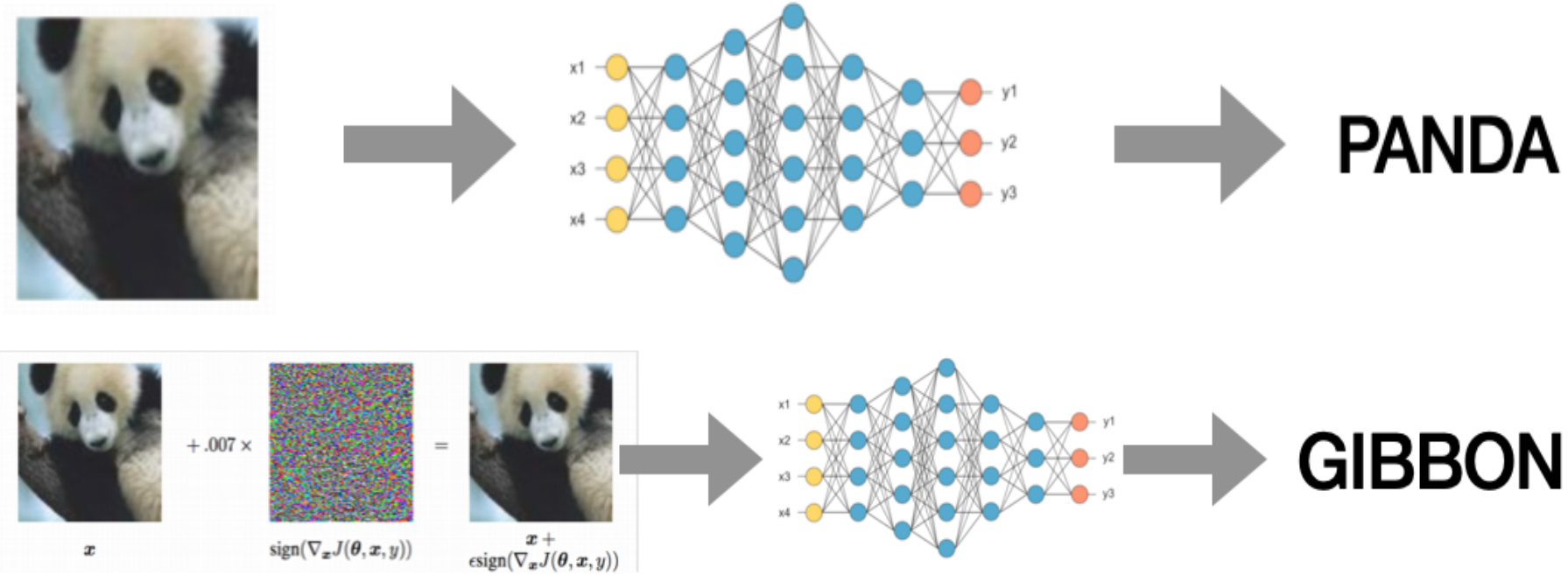
Robustness to adversarial disturbances

Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise



Robustness to adversarial disturbances

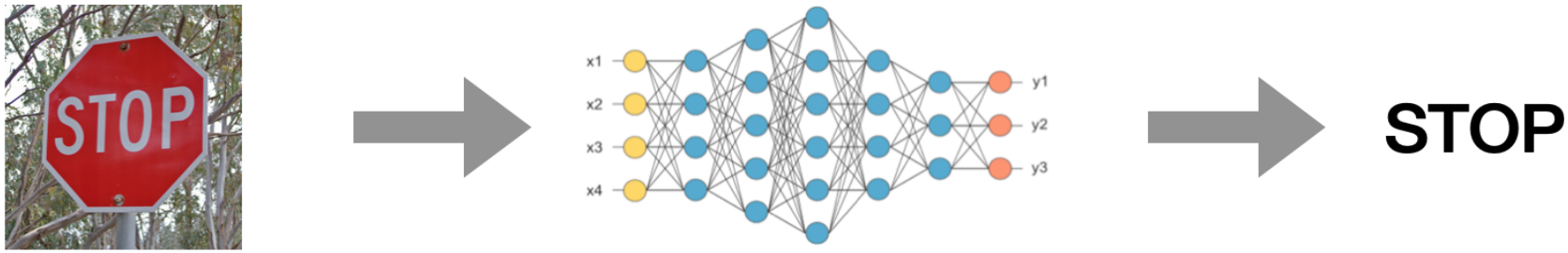
Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise



Source: Goodfellow et al, Explaining and Harnessing Adversarial Examples, 2015

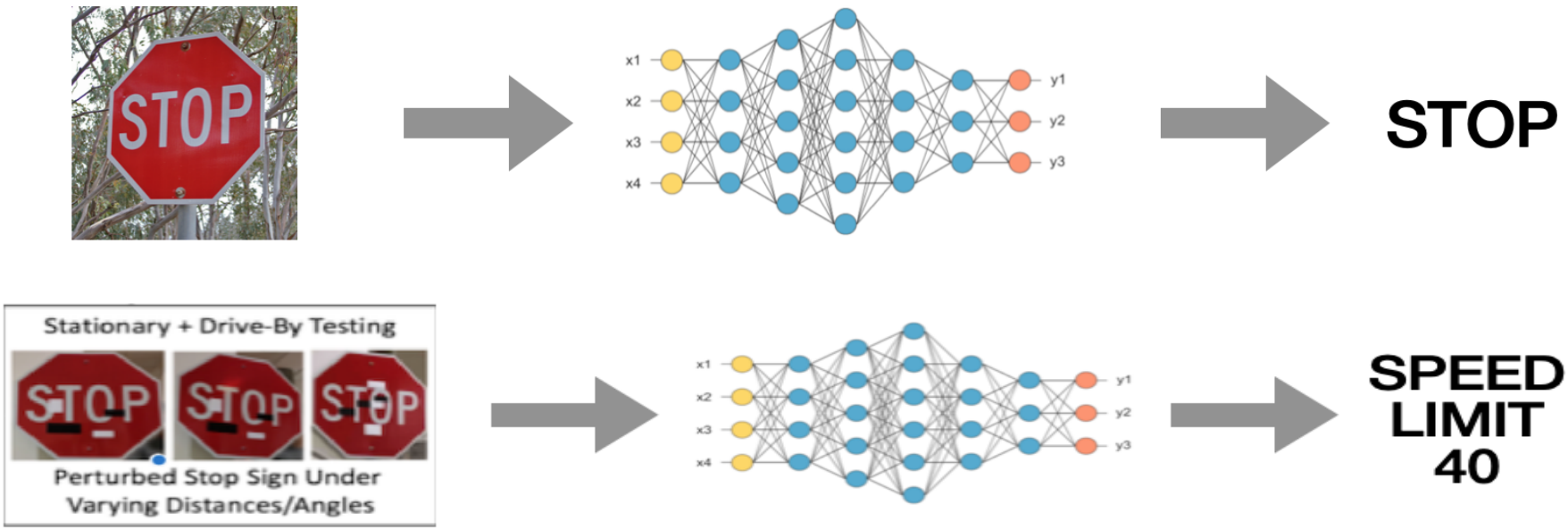
Robustness to adversarial disturbances

Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise



Robustness to adversarial disturbances

Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise



Source: Eykholt et al, Robust Physical-World Attacks on Deep Learning Visual Classification, 2018

Robustness to adversarial disturbances

Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise

If the NN has n outputs NN_1 to NN_n, the property that every image is classified to $i \in [1, n]$ writes:

$$\forall j \in [1, n], NN_i(x) \geq NN_j(x)$$

Local robustness

Small L_∞-norm perturbation does not affect classification (consider all images with values of each pixel at distance of up to ϵ from the corresponding pixel in the original image x^*)

$$\forall x, \|x - x^*\|_\infty \leq \epsilon \implies class(NN(x)) = class(NN(x^*))$$

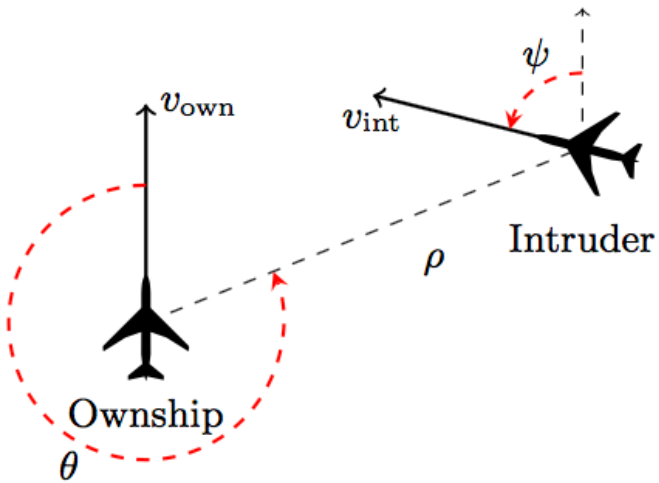
The verification problem is:

- find x in the neighborhood of x^* with fixed radius which does not satisfy this property or prove that there does not exist such an x,
- compute lower and upper bounds on maximal safety radius

Safety properties: Input-output relationships

Example of ACAS Xu: collision avoidance systems for civil aircrafts (FAA)

- New traffic alert and collision avoidance system : ACAS X (Airborne Collision Avoidance System X)
- Produces aicraft advisory (clear-of-conflict, weak right, weak left, strong right, etc.)
- Unmanned version : ACAS Xu, large lookup table of about 2GB.
- DNN representation proposed as a replacement, final version is an array of 45 DNNs (requiring about 3MB of memory)



ACAS Xu

Inputs

- ρ : Distance from ownship to intruder
- θ : Angle to intruder relative to ownship heading direction;
- ψ : Heading angle of intruder relative to ownship heading direction;
- v_{own} : Speed of ownship;
- v_{int} : Speed of intruder;
- τ : Time until loss of vertical separation;
- a_{prev} : Previous advisory.

Advisory (outputs)

- Clear-of-Conflict (COC), weak right, weak left, strong right, or strong left.

ACAS Xu representation by DNNs

45 DNNs

- Produced by discretizing τ and a_{prev} ; each one has 5 inputs (ρ, θ, ψ, v_{own} and v_{int}) and 5 outputs (score for COC, weak right, weak left, strong right, strong left).
- Each DNN is fully connected with 6 hidden layers (300 RELU nodes each DNN).

DNN outputs

- Clear-of-Conflict (COC), weak right, weak left, strong right, or strong left.
- The DNNs output a score for each action, the action with the lowest score is taken.

Safety verification of ACAS Xu

Reluplex : An Efficient SMT Solver for Verifying Deep Neural Networks, Katz et al, CAV 2017.

Some properties considered (under certain constraints)

- If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold, e.g.

$$\rho \geq 55947, v_{own} \geq 1145, v_{int} \leq 60 \implies \text{score of COC} \leq 1500$$

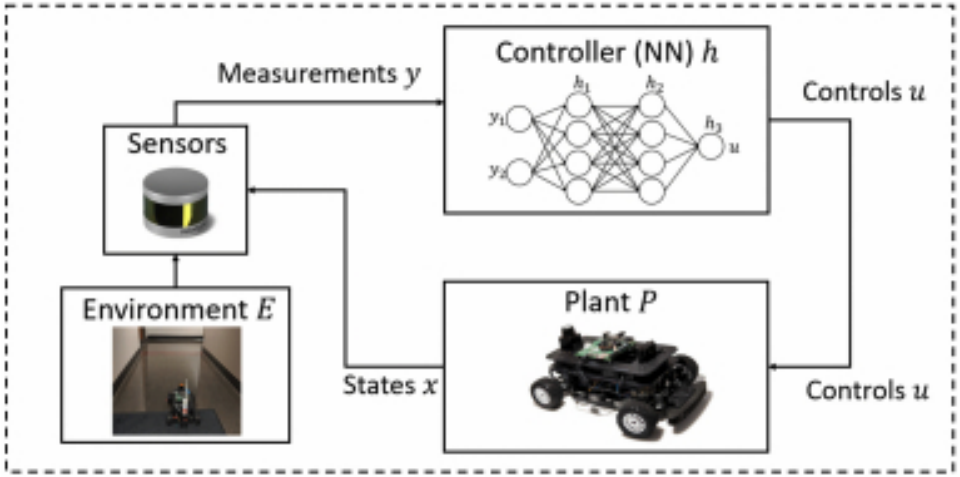
- If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- If the intruder is near and approaching from the left, network advises “strong right”.

Proof are needed, not just testing on some test cases. Supposes computing how sets of inputs are propagated.

Verifying the closed-loop system

Given

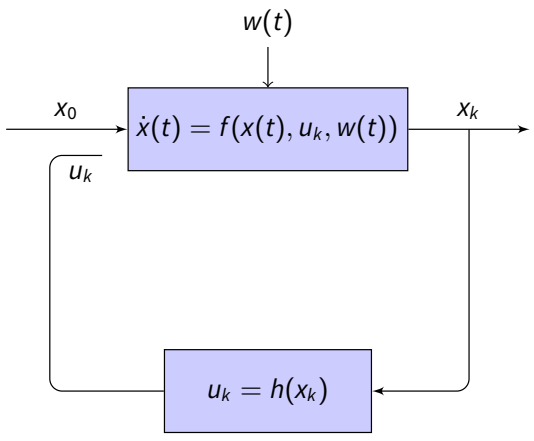
- plant dynamic f,
- state x, control u, disturbance $w \in W$
- NN controller h
- control period Δt_u

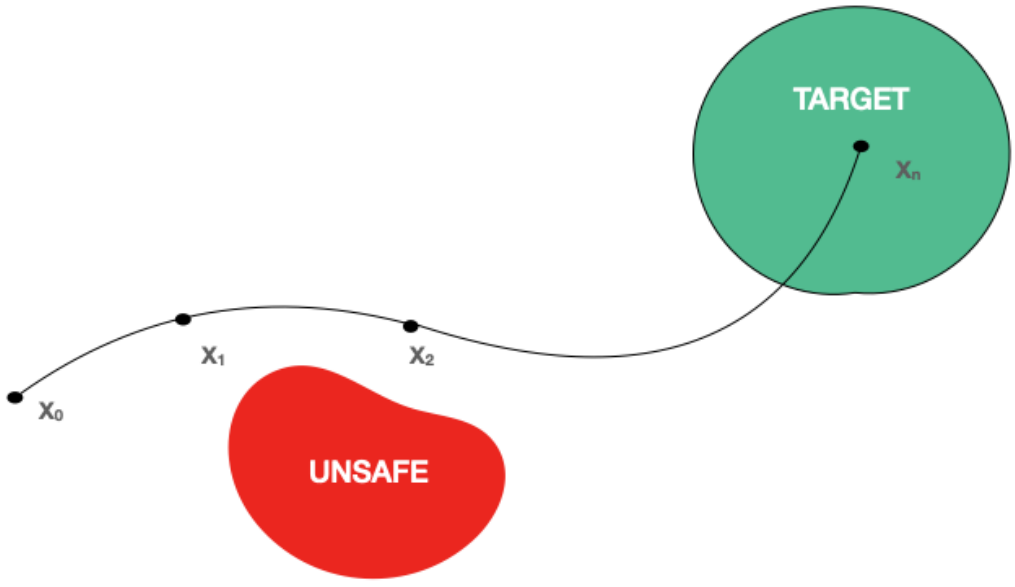


Composed system (plant + neural network controller) is a time-triggered (u computed every $\Delta_u t$) dynamical system with non-linear feedback:

$$
\begin{aligned}
\dot{x}(t) &= f(x(t), u(t), w(t)) \\
x(t_0) &= x_0 \in X_0 \\
u(t) &= u_k = h(y(x(\tau_k))), \text{ for } t \in [\tau_k, \tau_{k+1}), \text{ with } \tau_k = t_0 + k\Delta t_u, \ \forall k \geq 0
\end{aligned}
$$

Reachability analysis for safety and robustness verification

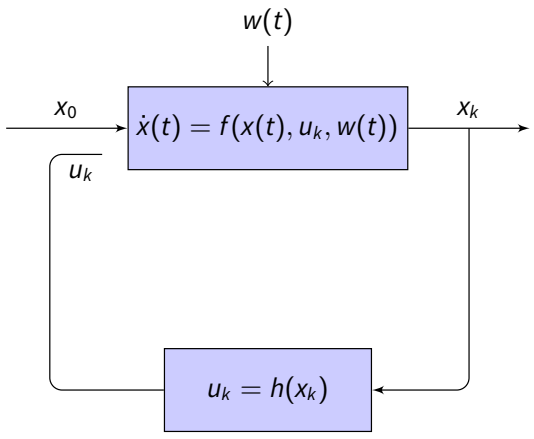


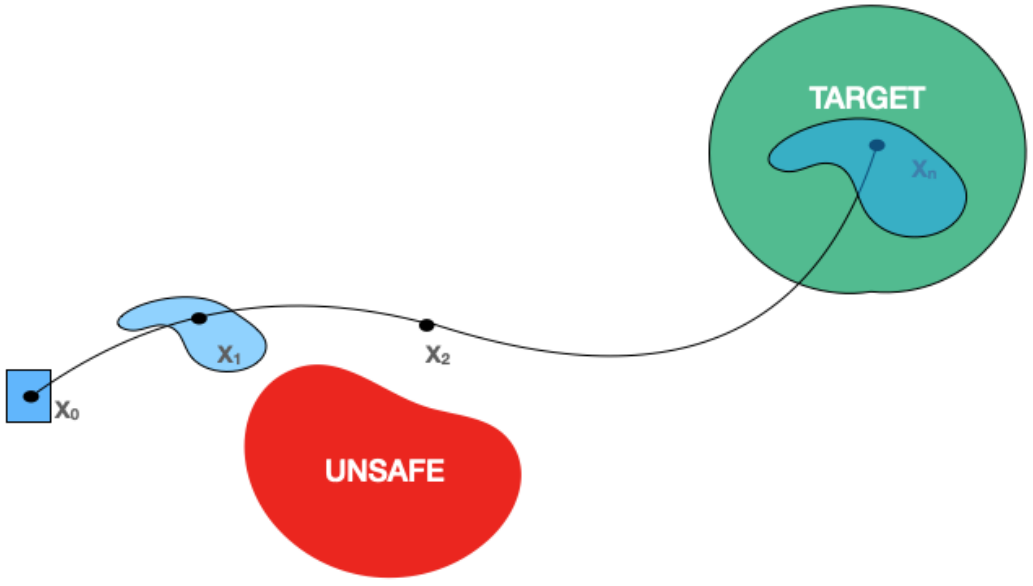


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions

Reachability analysis for safety and robustness verification

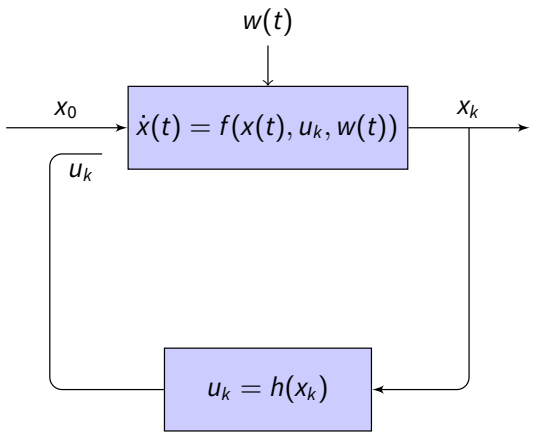


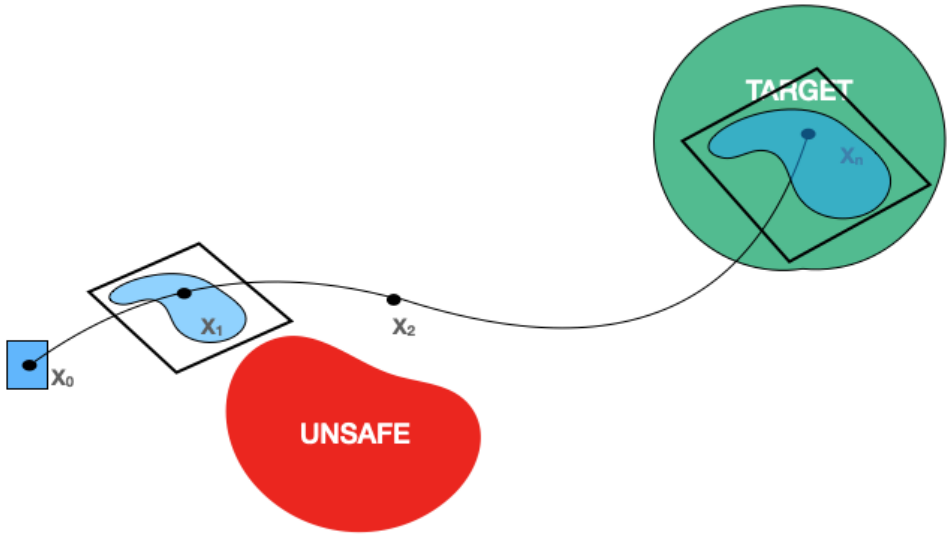


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- Also for noisy initial conditions x_0 (robustness)

Reachability analysis for safety and robustness verification

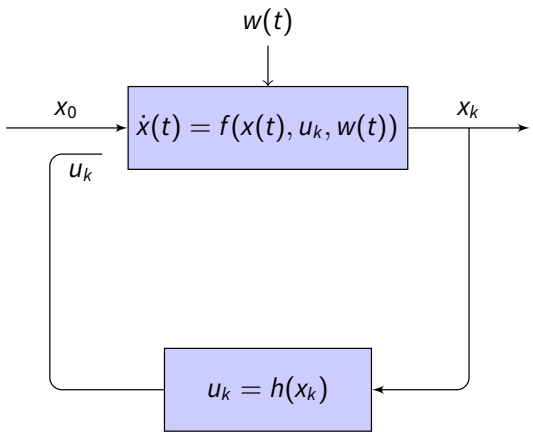


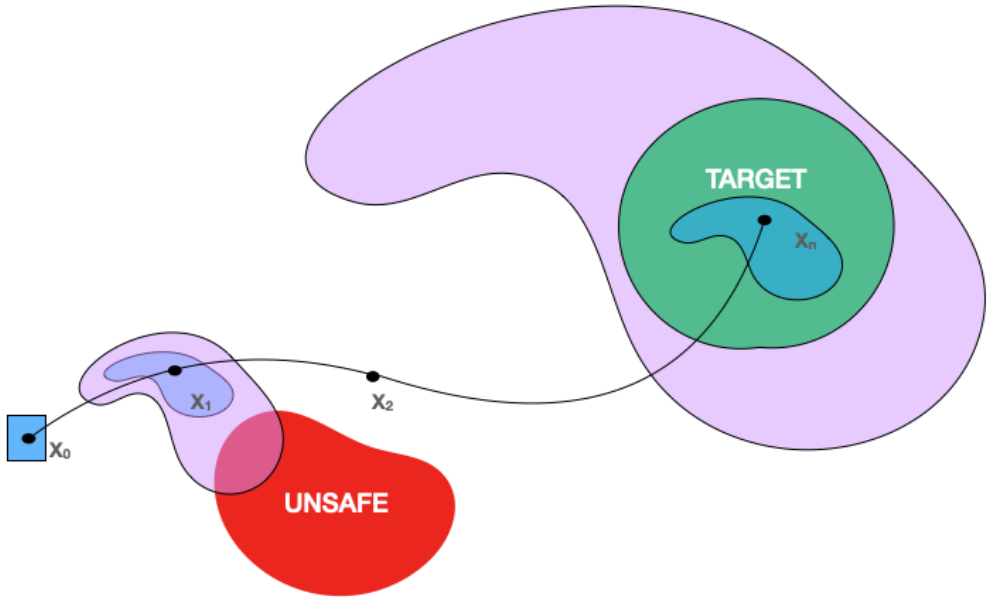


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- Also for noisy initial conditions x_0 (robustness)
  - Proven by over-approximated reachability

Reachability analysis for safety and robustness verification

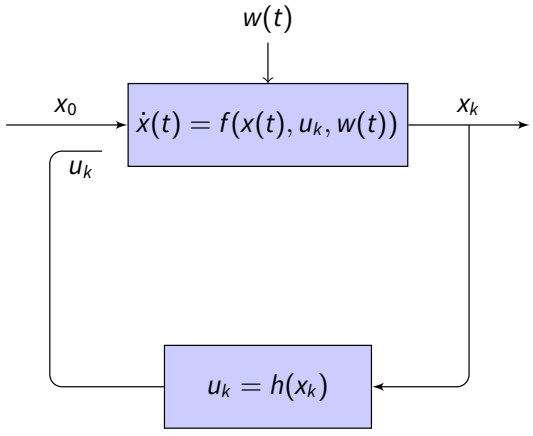


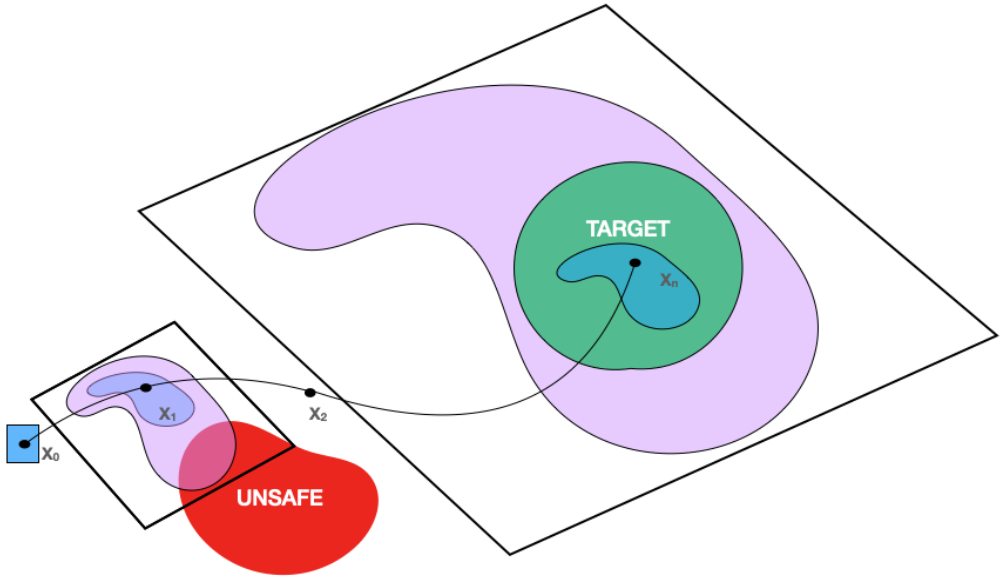


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- And external disturbances $w(t)$

Reachability analysis for safety and robustness verification

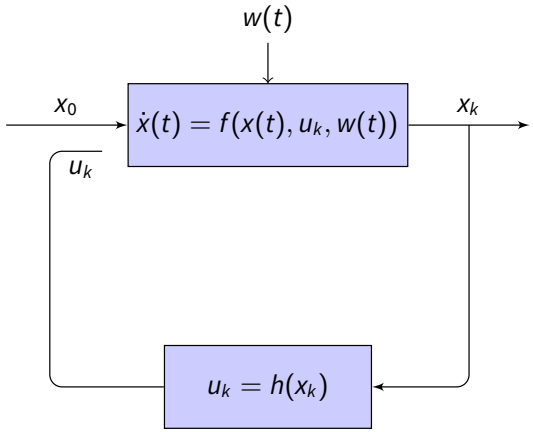


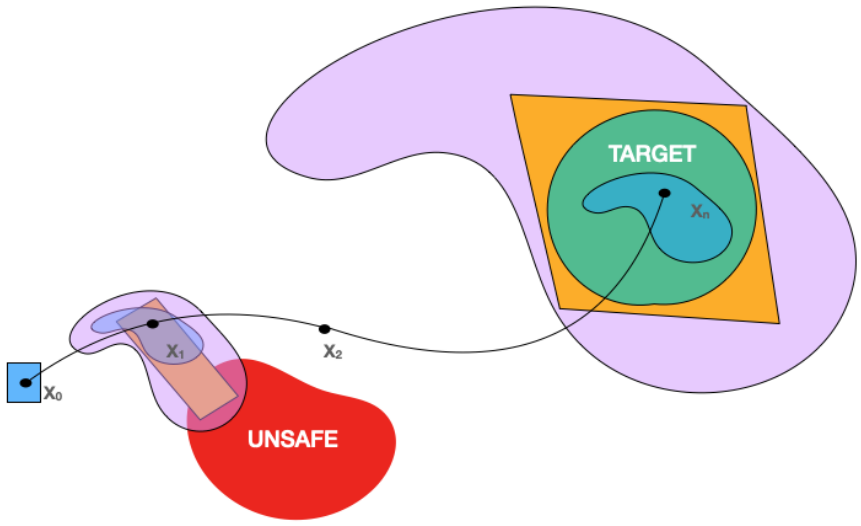


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- And external disturbances $w(t)$
  - (Maximal) over-approximation unconclusive

Reachability analysis for safety and robustness verification

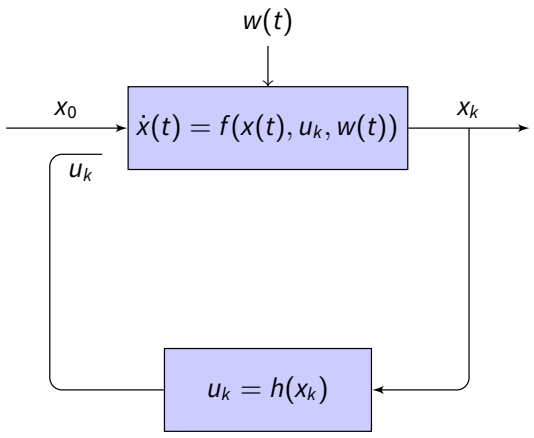


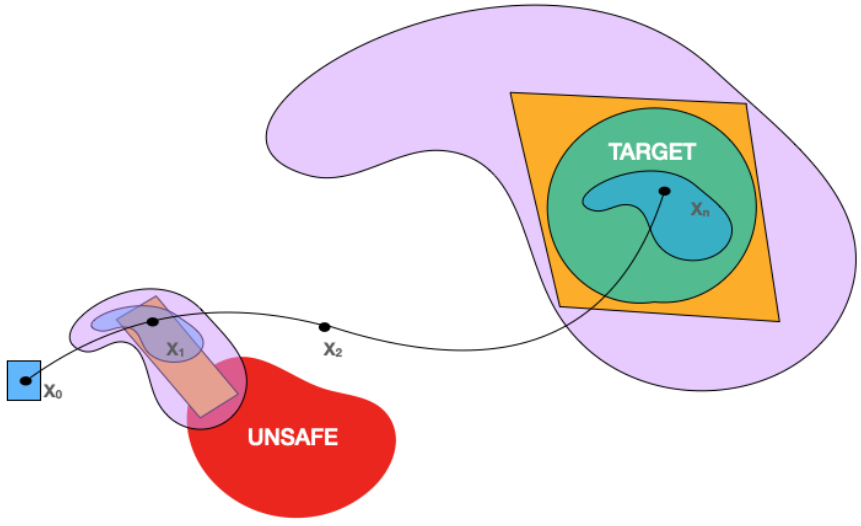


Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- And external disturbances $w(t)$
  - **Under-approximation**: $\exists x_0, \exists w(t)$ such that the trajectory is unsafe

Reachability analysis for safety and robustness verification





Over-approximate the sets of reachable states by flowpipes: prove bounds on max. distance to reference trajectory, obstacle avoidance, robustness, etc.

- Reach-Avoid problem: reaching target region while avoiding unsafe regions
- And external disturbances $w(t)$
  - **Under-approximation**: $\exists x_0, \exists w(t)$ such that the trajectory is unsafe
  - **Under-approximation**: $\forall x$ in target, $\exists x_0, \exists w(t)$ s.t. x is reached (target covered) + some final states proven to be outside the target

Reachability problems with disturbances w

Compute inner and outer-approximating sets $I(t)$ and $O(t)$ such that:

- Maximal reachability

$$I_{\mathcal{E}}(t) \subseteq R^{f,h}_{\mathcal{E}}(t; \mathbb{X}_0, \mathbb{W}) = \{x \,|\, \exists w \in \mathbb{W}, \exists x_0 \in \mathbb{X}_0,\ x = \varphi^{f,h}(t; x_0, w)\} \subseteq O_{\mathcal{E}}(t)$$

- Minimal or robust reachability

$$I_{\mathcal{AE}}(t) \subseteq R^{f,h}_{\mathcal{AE}}(t; \mathbb{X}_0, \mathbb{W}) = \{x \,|\, \forall w \in \mathbb{W}, \exists x_0 \in \mathbb{X}_0,\ x = \varphi^{f,h}(t; x_0, w)\} \subseteq O_{\mathcal{AE}}(t)$$

We have:

$$R^{f,h}_{\mathcal{AE}}(t; \mathbb{X}_0, \mathbb{W}) \subseteq R^{f,h}_{\mathcal{E}}(t; \mathbb{X}_0, \mathbb{W})$$

Illustration (lab session part 1)

Go to Section "Experimenting with existing analyses" (implemented in the Julia Neural Verification Library) of the Lab sheet 1 on moodle:

1. Verification of a partial specification for the ACAS Xu Networks
2. Robustness of an Image Classification Network: formal methods and adversarial counter-examples
3. (Neural network controlled systems will be seen later: Lecture 4)

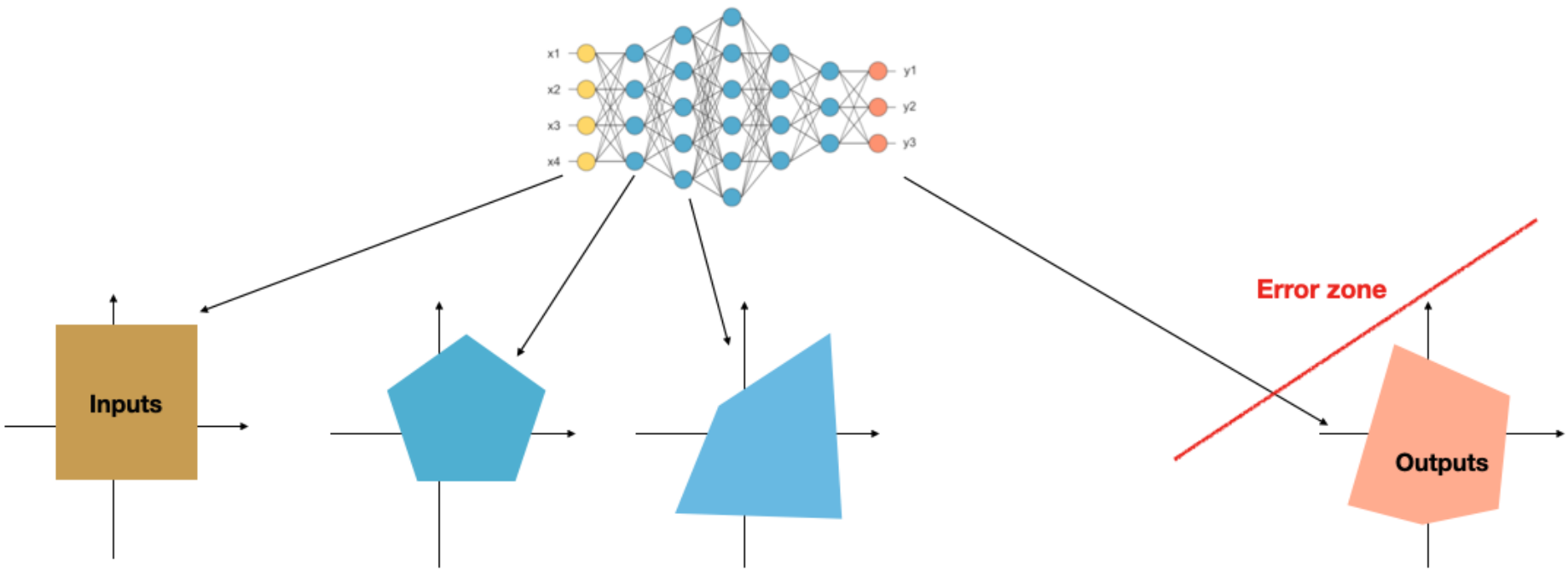
Note that the principles of the Reluplex approach, used as the solver in notebooks 1 and 2, will be described in Lecture 3

Reluplex: An Efficient SMT Solver for Verifying Deep Neural Network" G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer (International Conference on Computer Aided Verification, 2017

Reachability Analysis for Neural Network Verification

All these properties:

- Need to be proved for (possibly large) sets of network inputs
- Can be specified as preconditions/postconditions expressed in linear arithmetic
- Can be proved or falsified with different approaches (constraint-based, abstraction-based)



Outline

Introduction to the verification of neural networks

1. Introduction to reliable AI in autonomous CPS
2. Abstraction-Based Verification (Lectures 1 and 2)
3. Constraint-Based Verification (Lecture 3)
4. Application to neural network controlled systems (Lecture 4)

A recent and quickly evolving field, but heavily relying on techniques from traditional program and systems verification.

Propagating sets through NN: the Box abstraction

Box (or Hyperrectangle)

For a vector of variables $x \in \mathbb{R}^n$, a Box is a Cartesian product of n Intervals $[a_i, b_i] = \{x_i \in \mathbb{R}, x_i \geq a_i \wedge x_i \leq b_i\}$ for each x_i, $i = 1, \ldots, n$.

Abstract transformers on Intervals

For $a, b, c, d \in \mathbb{R}^n$ and $\lambda > 0$:

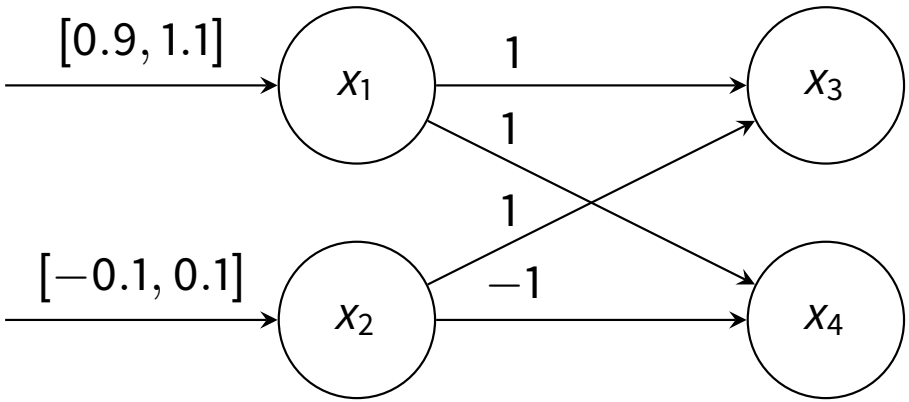
$$
\begin{aligned}
[a, b] +^{\#} [c, d] &= [a + c, b + d] \\
-^{\#} [a, b] &= [-b, -a] \\
ReLU^{\#}[a, b] &= [ReLU(a), ReLU(b)] \\
\lambda^{\#}[a, b] &= [\lambda a, \lambda b]
\end{aligned}
$$

- The abstract transformers must be a sound over-approximation of the corresponding concrete transformers: e.g.

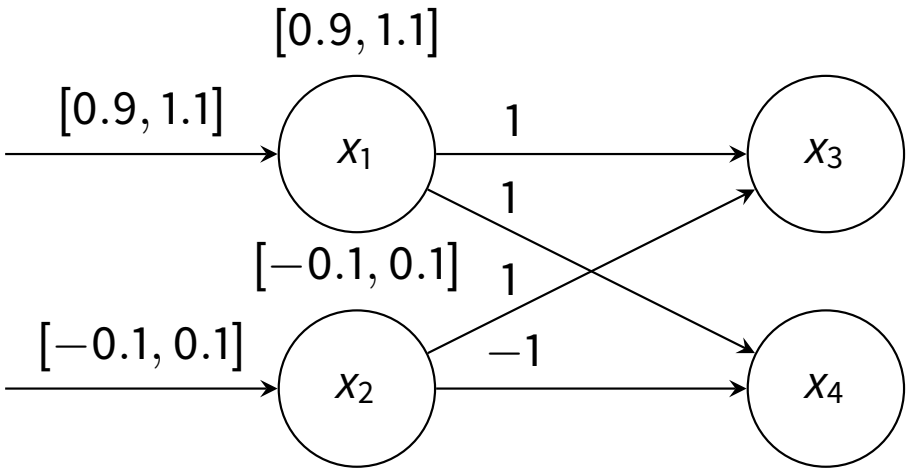
$$\{x + y,\ x \in [a, b] \wedge y \in [c, d]\} \subseteq [a, b] +^{\#} [c, d] = [a + c, b + d]$$

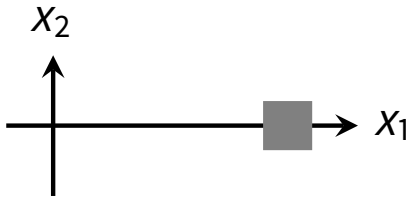
- The Box abstraction is non relational: it cannot capture relations between dimensions

Example: analysis of a single layer

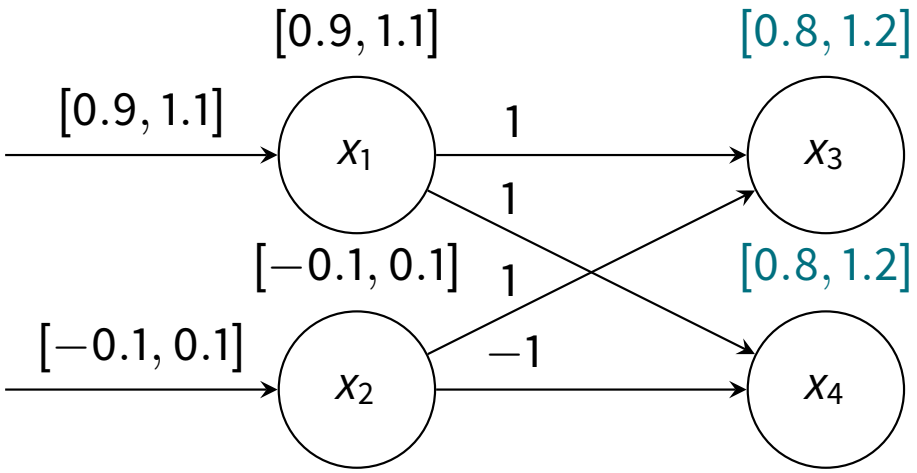


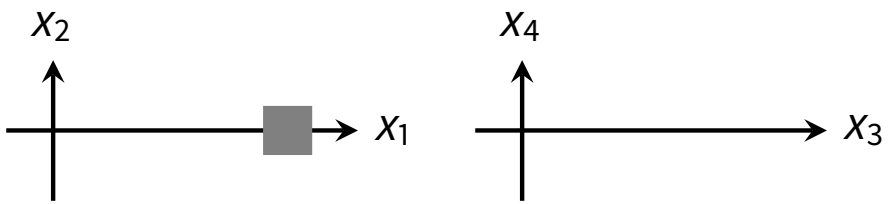
Example: analysis of a single layer





Example: analysis of a single layer





Domain using Boxes:

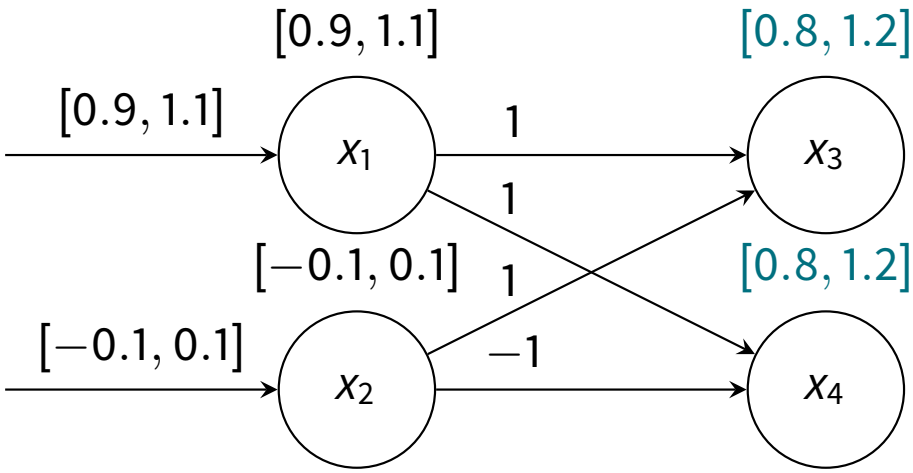
$$0.9 \leq x_1 \leq 1.1$$

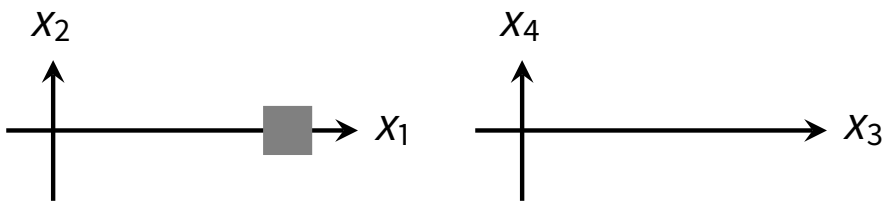
$$-0.1 \leq x_2 \leq 0.1$$

$$0.8 \leq x_3 \leq 1.2$$

$$0.8 \leq x_4 \leq 1.2$$

Example: analysis of a single layer





Domain using Boxes:

$$0.9 \leq x_1 \leq 1.1$$

$$-0.1 \leq x_2 \leq 0.1$$

$$0.8 \leq x_3 \leq 1.2$$

$$0.8 \leq x_4 \leq 1.2$$

Exact domain is:

$$0.9 \leq x_1 \leq 1.1$$

$$-0.1 \leq x_2 \leq 0.1$$

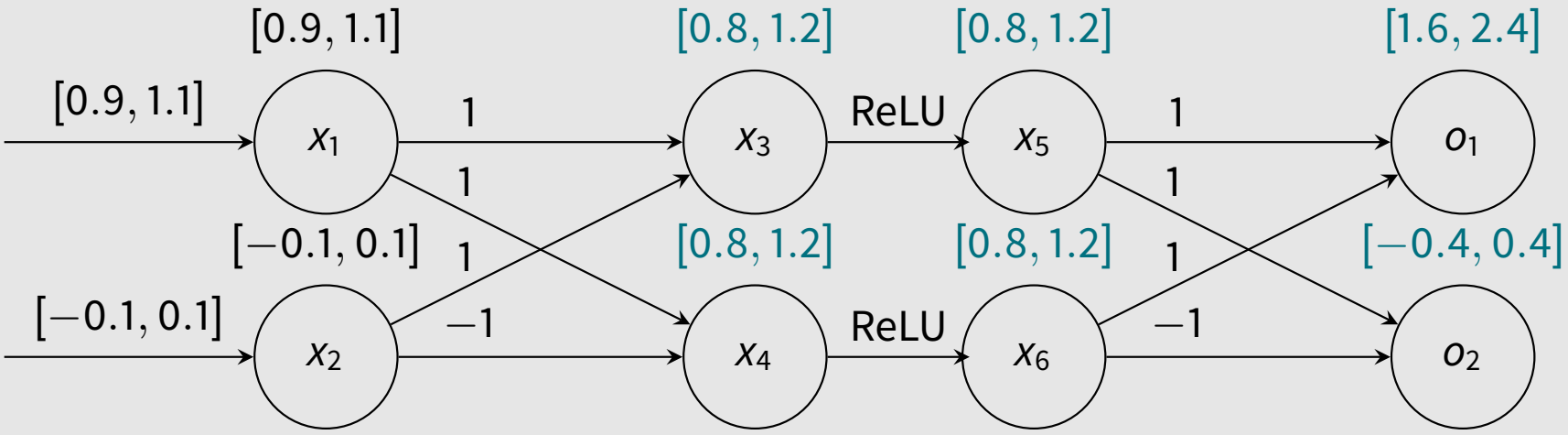
$$x_3 = x_1 + x_2$$

$$x_4 = x_1 - x_2$$

The optimal box transformers are not exact.

The Box abstraction can prove some properties

Example (Verifying Robustness)

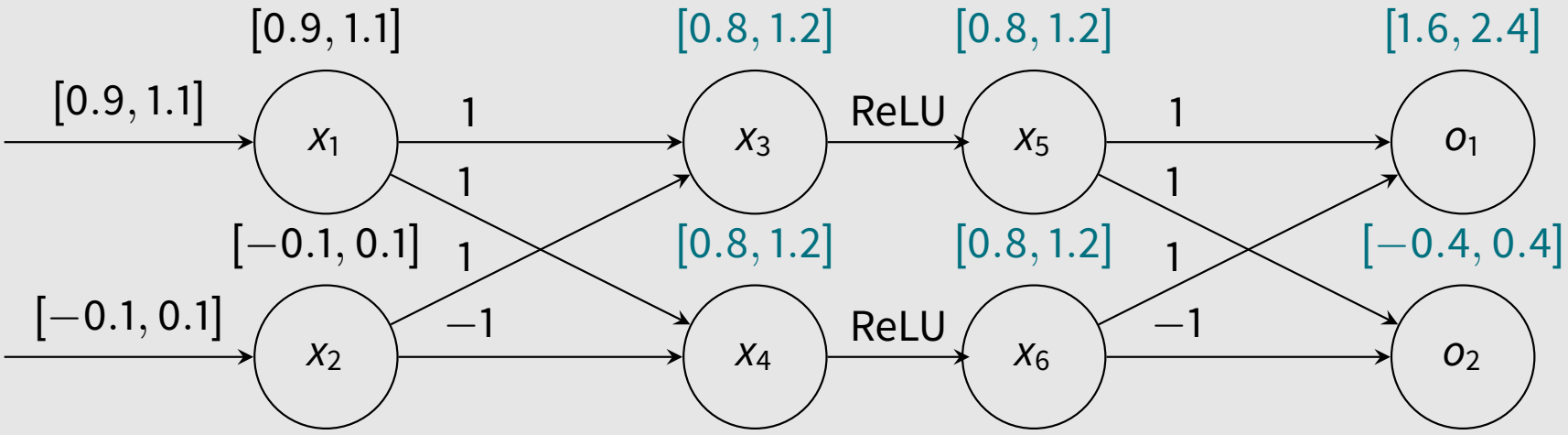


Robustness problem:

- for a given input vector $x = (x_1, x_2)$ (think of the pixel values of an image), the result is classified by the network with class 1 if $o_1(x) \geq o_2(x)$, otherwise 2.
- do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.1 on all components ?

The Box abstraction can prove some properties

Example (Verifying Robustness)



Robustness problem:

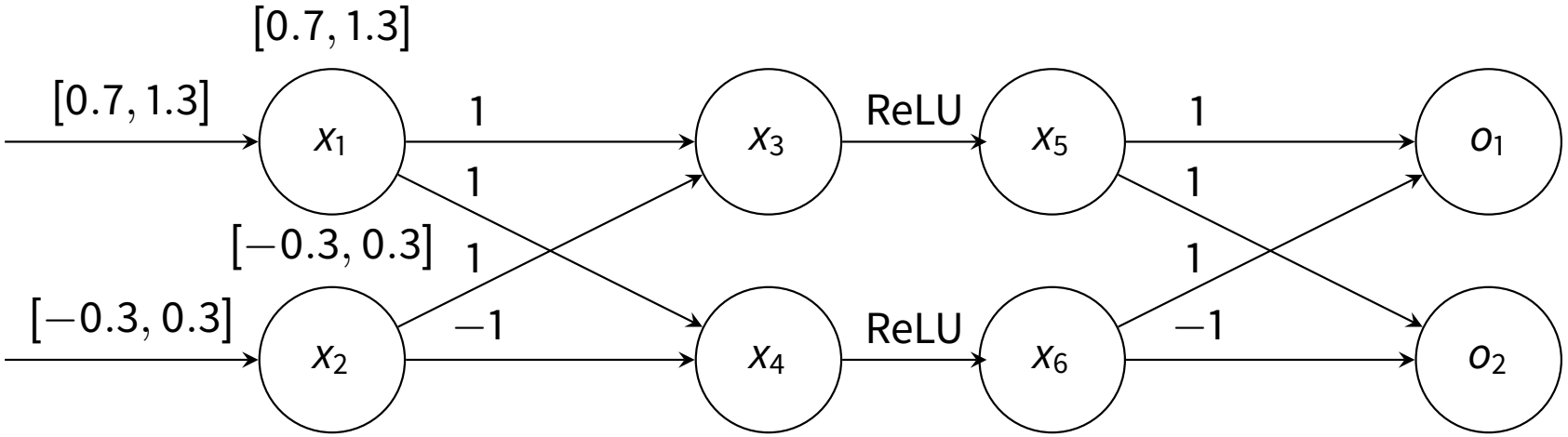
- for a given input vector $x = (x_1, x_2)$ (think of the pixel values of an image), the result is classified by the network with class 1 if $o_1(x) \geq o_2(x)$, otherwise 2.
- do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.1 on all components ?
- YES: $\forall o_1 \in [1.6, 2.4], o_2 \in [-0.4, 0.4]$, we always have $o_1(x) \geq o_2(x)$.

But Boxes can also fail to prove some true properties

Do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.3 on all components ?

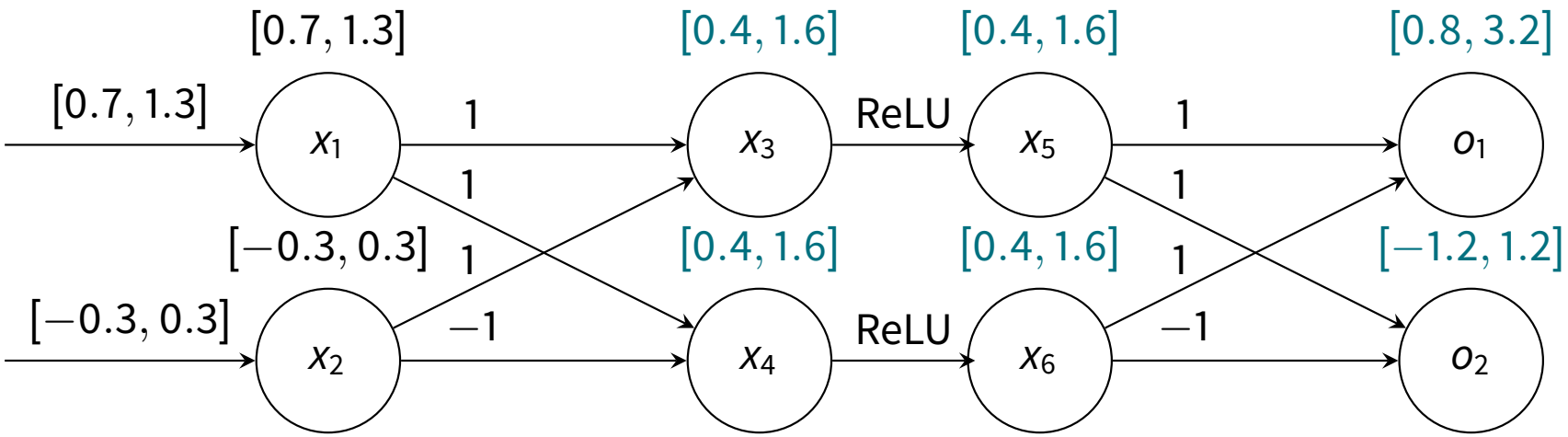
But Boxes can also fail to prove some true properties

Do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.3 on all components ?



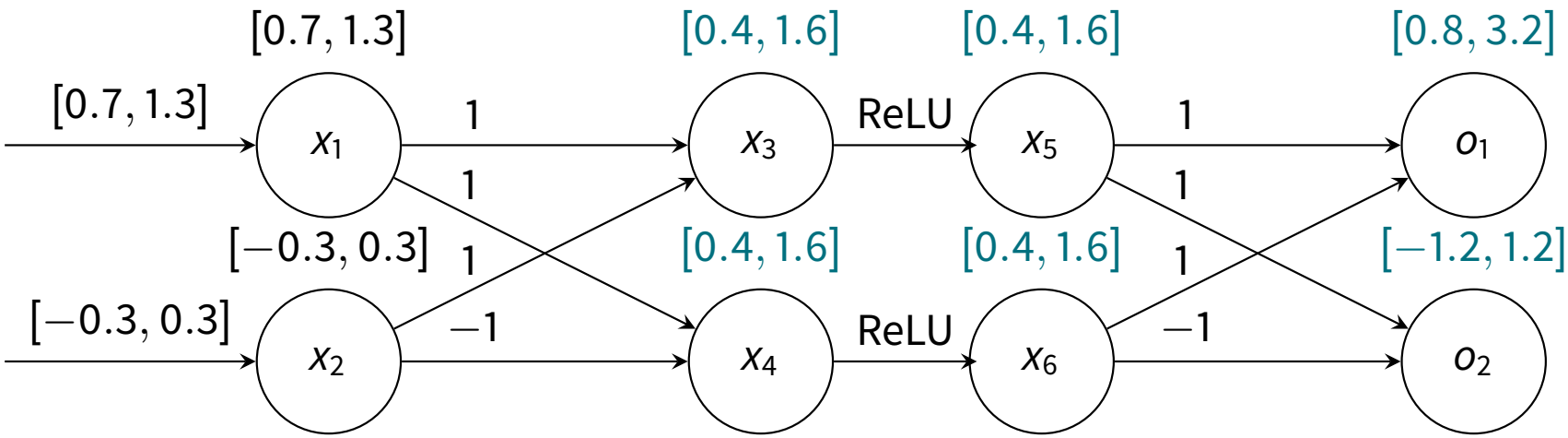
But Boxes can also fail to prove some true properties

Do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.3 on all components ?



But Boxes can also fail to prove some true properties

Do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.3 on all components ?



- there is a non empty intersection between the range for o_1 and o_2: boxes cannot conclude about the robustness
- while exact analysis allows to conclude that the decision is robust:

$$0.7 \leq x_1 \leq 1.3 \qquad x_5 = x_3 = x_1 + x_2 \qquad o_1 = x_5 + x_6 = 2x_1$$

$$-0.3 \leq x_2 \leq 0.3 \qquad x_6 = x_4 = x_1 - x_2 \qquad o_2 = x_5 - x_6 = 2x_2$$

By noting that $x_1 \geq x2$, we can conclude $o_1 \geq o_2$.

Abstraction-based verification of neural network

An application of Abstract Interpretation for automatic program verification

- Abstract interpretation is a theory of semantics approximation (Patrick and Radhia Cousot, 1977)
- Provides a systematic way to build automated program analyzers
- By synthesizing local abstract invariants fully automatically, from the source code

Yields incomplete but scalable methods:

- Abstract program semantics in a conservative way with efficient computation
  - Compute sound enveloppes of state variables
  - Bound and propagate uncertainties
- A branch of abstractions includes the numerical abstract domains: convex abstractions such as Boxes, Zonotopes and Polyhedra

Automatic program verification?

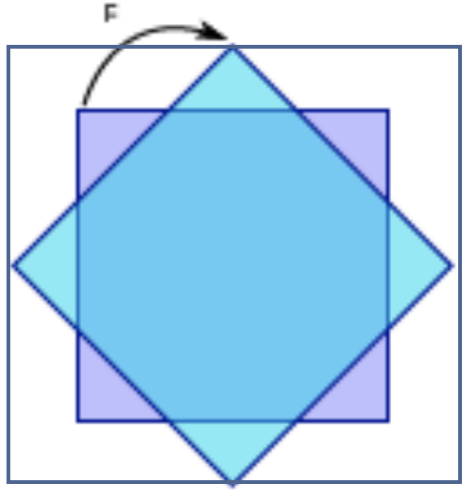
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



Automatic program verification?

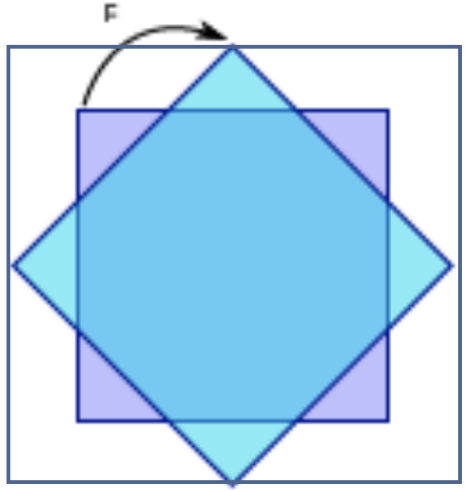
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x :=Input[-1,1];
y :=Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?

Automatic program verification?

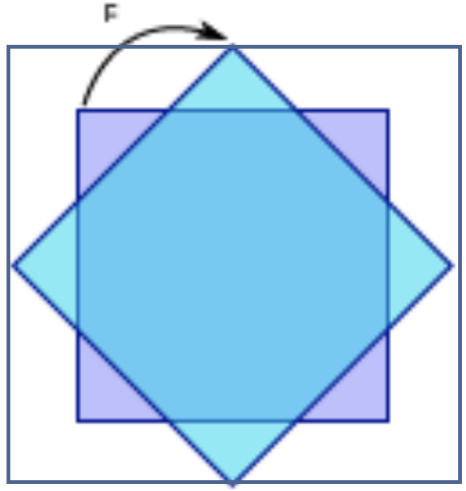
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x :=Input[-1,1];
y :=Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)

Automatic program verification?

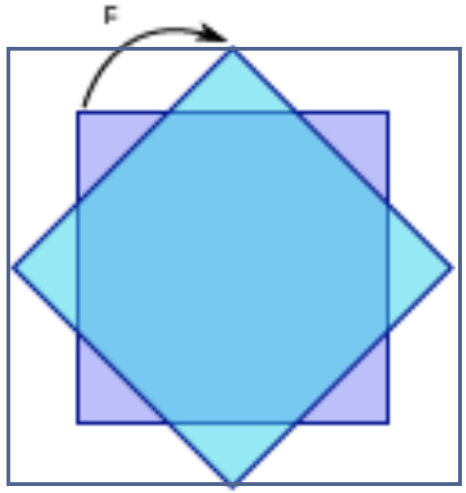
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?

Automatic program verification?

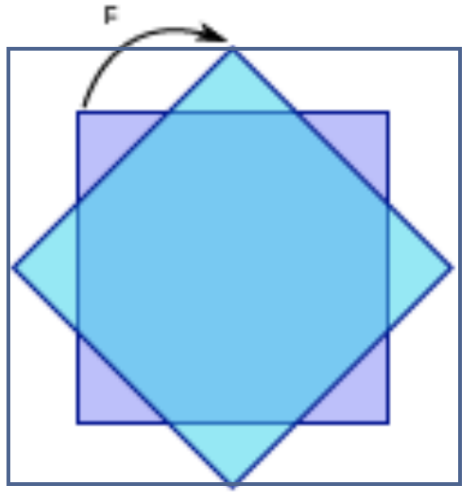
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x :=Input[-1,1];
y :=Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?
  - Easy if G is an inductive invariant (G holds initially and $F(G) \subseteq G$), but not the case here

Automatic program verification?

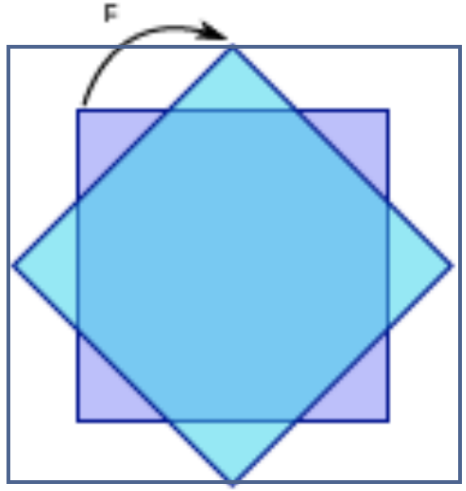
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?
  - Easy if G is an inductive invariant (G holds initially and $F(G) \subseteq G$), but not the case here
- And what if I have no idea what the invariant is ?

Automatic program verification?

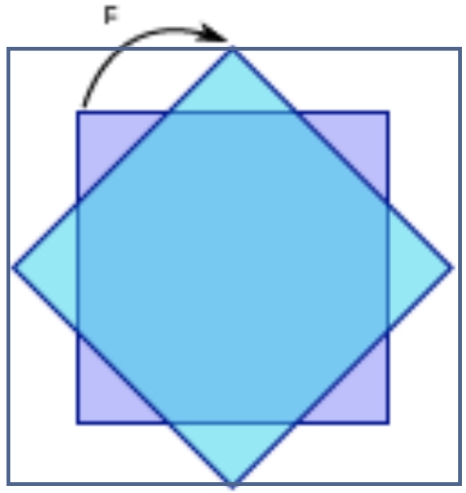
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?
  - Easy if G is an inductive invariant (G holds initially and $F(G) \subseteq G$), but not the case here
- And what if I have no idea what the invariant is ?

$\Rightarrow$ Synthesize automatically inductive invariants

Invariant synthesis

We are interested in local numerical invariants = at each program point, properties/values of variables true for all execution traces, and if possible the strongest properties: collecting semantics

Example

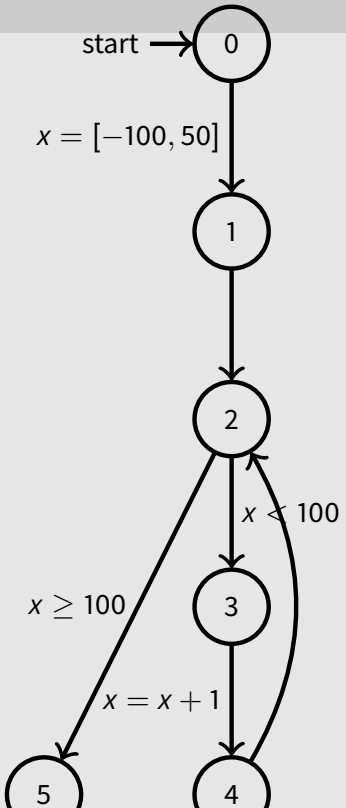
```
void main() {
  int x := {-100,-99,...,50};
  // X = {-100,-99,...,50}
  while (x<100) {
    // X = {-100,-99,...,99}
    x=x+1;
    // X = {-99,-98,...,100}
  }
   // X = {100}
}
```

Forward collecting semantics - informally

- We construct the control flow graph of a program, as a transition system $\mathcal{T} = (S, i, E, Tran)$
  - S is the set of control points of the program (we define [i] as the control point corresponding to the execution point between line i and line i + 1)
  - i is the initial control point
  - $E = \{x = expr \mid Aexp\} \cup Bexp$
  - We define a certain number of rules for the transitions

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1; // [4]
 }          // [5]
}
```



Forward collecting semantics - informally

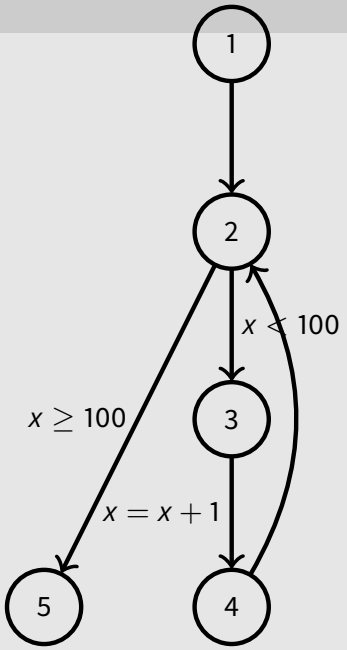
- We associate to each control point s_i of $\mathcal{T}$ an "equation" in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

where $[\![t]\!]$ is the interpretation of the transition t (/transfer function), seen as a function from the set of values of variables to itself

Example

```
void main()
{ // [0]
  int x=[-100,50];//[1]
  while /*[2]*/(x<100)
  {          // [3]
    x=x+1;   // [4]
  }          // [5]
}
```



$$\begin{aligned} x_0 &= \top \\ x_1 &= \{-100, \ldots, 50\} \\ x_2 &= x_1 \cup x_4 \\ x_3 &=]-\infty, 99] \cap x_2 \\ x_4 &= x_3 + 1 \\ x_5 &= [100, +\infty[\cap x_2 \end{aligned}$$

Forward collecting semantics - informally

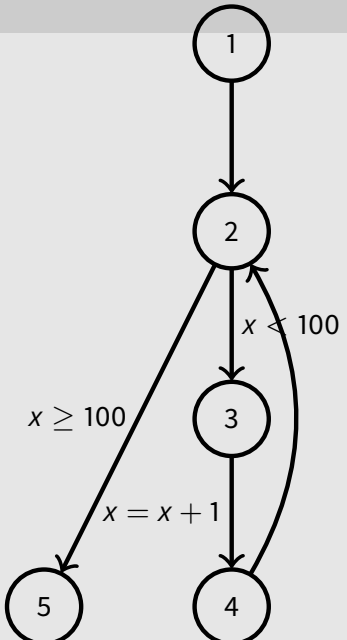
- We associate to each control point s_i of $\mathcal{T}$ an "equation" in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

- The solutions of this system $X = F(X)$ are the local invariants at control points s_i ("property" always true for all possible executions)
  - interested in the strongest properties = the smallest fixpoint

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1; // [4]
 }          // [5]
}
```



$$\begin{aligned} x_0 &= \top \\ x_1 &= \{-100, \ldots, 50\} \\ x_2 &= x_1 \cup x_4 \\ x_3 &=]-\infty, 99] \cap x_2 \\ x_4 &= x_3 + 1 \\ x_5 &= [100, +\infty[\cap x_2 \end{aligned}$$

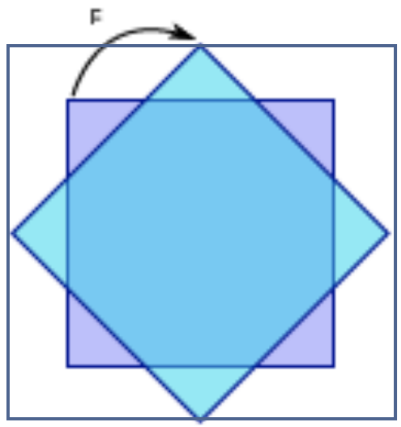
Synthesis of an abstract inductive invariant, informally

Abstract the forward collecting semantics and synthesize an inductive invariant

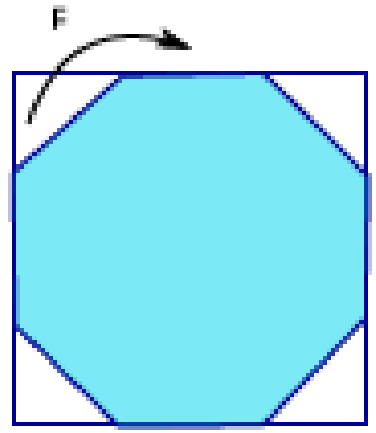
- the program semantics is abstracted by a transition relation or discrete dynamical system $F^\sharp$ that covers all possible behaviors, and can be efficiently computed
- we collect and gather all states that can be reached after zero, one, or any number of iteration steps of $F^\sharp$ = iterate $F^\sharp$ until fixpoint: Kleene iteration
- the least fixpoint when it exists is the strongest inductive invariant (S such that $I \subseteq S$ and $F^\sharp(S) \subseteq S$) of the abstract system

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

$x, y \in [-2, 2]$ loop invariant ?



no box is an inductive invariant



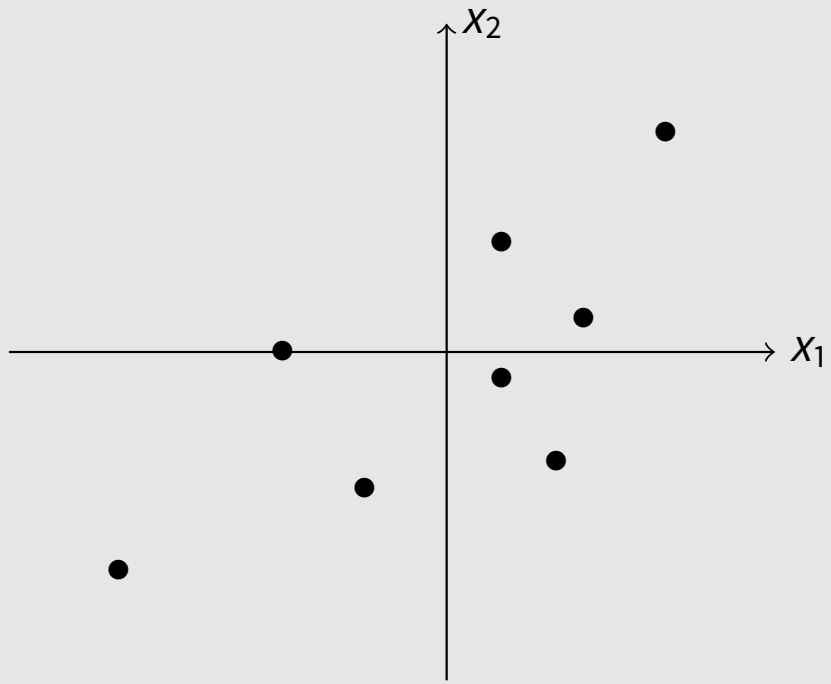
but we can find an inductive octagon

Art of abstract interpretation= define a good general-purpose abstraction!

Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

Example (Concrete points,)

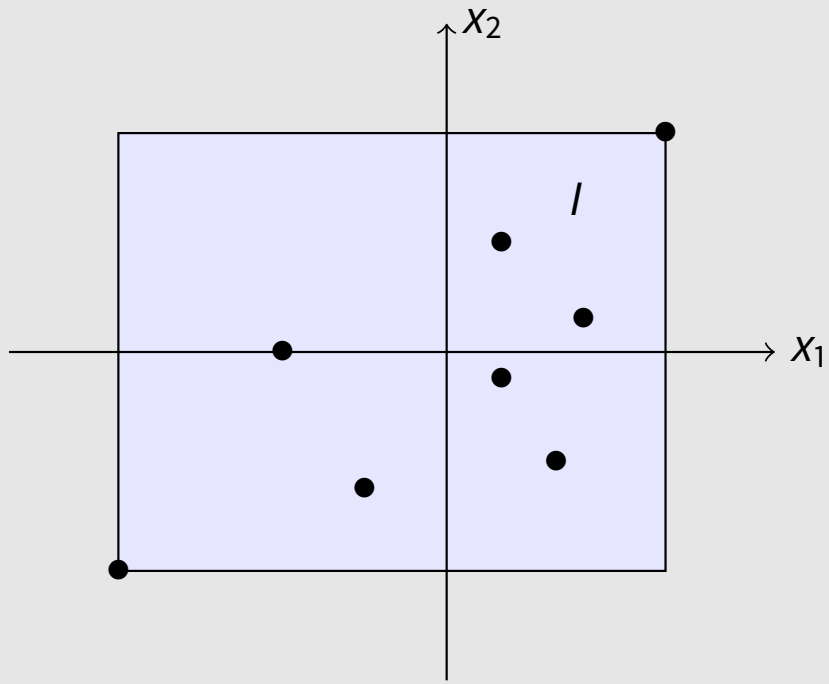


Abstract states over-approximate sets of concrete states: we reason in the abstract to deduce properties true on the concrete executions

Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

Example (Concrete points, Interval)

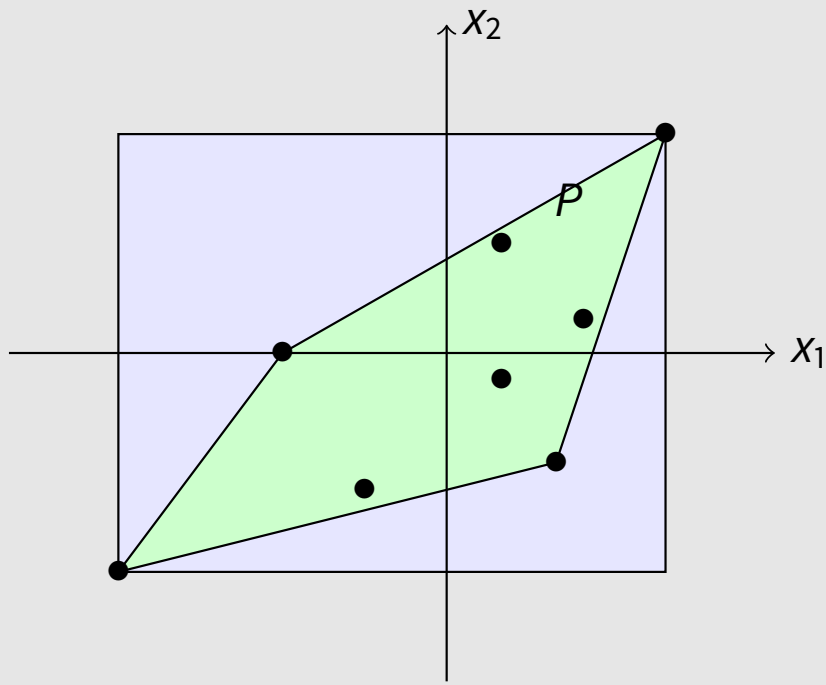


Abstract states over-approximate sets of concrete states: we reason in the abstract to deduce properties true on the concrete executions

Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

Example (Concrete points, Interval and Polyhedral abstractions)



Abstract states over-approximate sets of concrete states: we reason in the abstract to deduce properties true on the concrete executions

Abstract Interpretation (Cousot & Cousot 77)

Abstract interpretation: a theory of semantics approximation

- The simple/elegant framework relies on Galois connections between complete lattices (abstraction by intervals for instance)
- Weaker structures often used in practice

Galois connections

- Let $\mathcal{C}$ be a complete lattice of concrete properties: (e.g. $(\wp(\boldsymbol{Z}), \subseteq)$)
- Let $\mathcal{A}$ be a complete lattice of abstract properties: (e.g. $(S, \sqsubseteq)$)
- $\alpha : \mathcal{C} \to \mathcal{A}$ (abstraction) and $\gamma : \mathcal{A} \to \mathcal{C}$ (concretisation) two monotonic functions (we could forget this hypothesis) such that

$$\alpha(x) \leq_{\mathcal{A}} y \Leftrightarrow x \leq_{\mathcal{C}} \gamma(y)$$

Abstract states over-approximate sets of concrete states: we will reason in the abstract to deduce properties true on the concrete executions

Abstraction

- The (best) abstraction function α gives the most precise abstract value that soundly represents a given concrete property
- The concretization function γ gives the semantics of abstract values, in terms of concrete properties: it maps an abstract value *a* to the largest set of concrete values that satisfy *a*

Not all abstractions are Galois connections

- There can be no α
- Example: convex polyhedra (no best abstraction of a disk)

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Complete:

$$\bigcup_{i \in I} [a_i, b_i] = [inf_{i \in I}\, a_i, sup_{i \in I}\, b_i]$$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

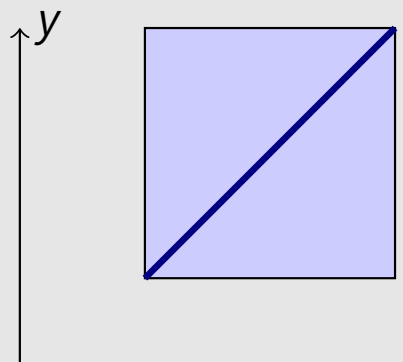
- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Complete:

$$\bigcup_{i \in I} [a_i, b_i] = [inf_{i \in I}\, a_i, sup_{i \in I}\, b_i]$$

Intervals are a non-relational abstraction: we forget relations between variables

```
int x,y; x=2+rand(0,2);
y=x; x-x = ?
```



Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{lcll} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & \end{array}$$

Concretisation

$$\begin{array}{lcll} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a,b] & \to & \end{array}$$

(a and b are potentially infinite)

Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{rcccl} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & [\mathit{inf}\ S, \mathit{sup}\ S] \end{array}$$

(the *inf* and *sup* are taken in $\mathbb{R} \cup \{-\infty, \infty\}$)

Concretisation

$$\begin{array}{rcccl} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a, b] & \to & \end{array}$$

(a and b are potentially infinite)

Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{rcccl} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & [\mathit{inf}\ S, \mathit{sup}\ S] \end{array}$$

(the *inf* and *sup* are taken in $\mathbb{R} \cup \{-\infty, \infty\}$)

Concretisation

$$\begin{array}{rcccl} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a, b] & \to & \{x \in \mathbb{R} \mid a \leq x \leq b\} \end{array}$$

(a and b are potentially infinite)

Abstraction of execution steps

Best abstract transformer of a transfer function F

For all concrete functionals $F : \mathcal{C} \to \mathcal{C}$, we define an abstract functional $F^\sharp : \mathcal{A} \to \mathcal{A}$ by

$$F^\sharp(y) = \alpha \circ F \circ \gamma(y)$$

It is the best possible abstraction of F.

Sound abstract transformer

We can use any over-approximation F' such that $F^\sharp(y) \leq_{\mathcal{A}} F'(y)$:

- must always be conservative (not lose any behavior)
- should provide a good balance between cost and precision

Transfer functions for arithmetic expressions

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

Transfer functions for arithmetic expressions

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

We compute:

$$\begin{aligned}
[a,b] \oplus [c,d] &= \alpha(\gamma([a,b]) + \gamma([c,d])) \\
&= \alpha(\{x \mid a \leq x \leq b\} + \{y \mid c \leq y \leq d\}) \\
&= \alpha(\{x+y \mid a \leq x \leq b,\ c \leq y \leq d\}) \\
&= \alpha(\{x+y \mid a+c \leq x \leq b+d\}) \\
&= [a+c, b+d]
\end{aligned}$$

(+ is extended to infinite numbers)

Transfer functions for arithmetic expressions

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

We compute:

$$\begin{aligned}
[a,b] \oplus [c,d] &= \alpha(\gamma([a,b]) + \gamma([c,d])) \\
&= \alpha(\{x \mid a \leq x \leq b\} + \{y \mid c \leq y \leq d\}) \\
&= \alpha(\{x+y \mid a \leq x \leq b,\ c \leq y \leq d\}) \\
&= \alpha(\{x+y \mid a+c \leq x \leq b+d\}) \\
&= [a+c, b+d]
\end{aligned}$$

(+ is extended to infinite numbers)

Exercise: what if one of the arguments is $\bot$, $\top$? What about multiplication/division?

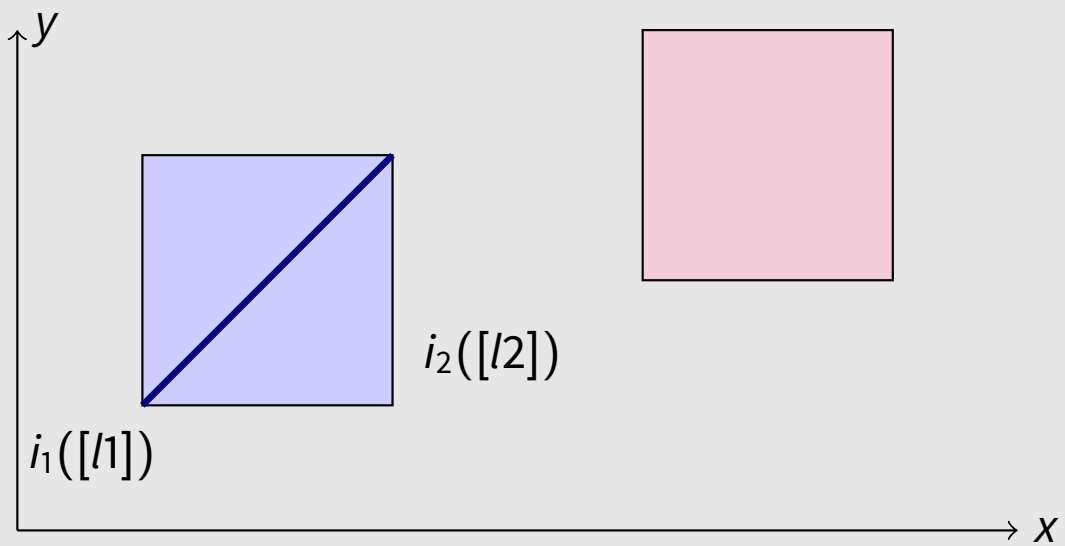
Abstraction and control flow joins

In case of conditional structures

- We must over-approximate the union of abstract values computed in the different branches
- We may lose some precision, but not forget behaviors

Example (Intervals)

```
int x,y;
if (random(0,1)<0.5) {
  x=2+rand(0,2);
  y=x;  [l1]  }
else {   x=6+rand(0,2) ;
  y=3+rand(0,2); [l2]
  }
[l3]
```



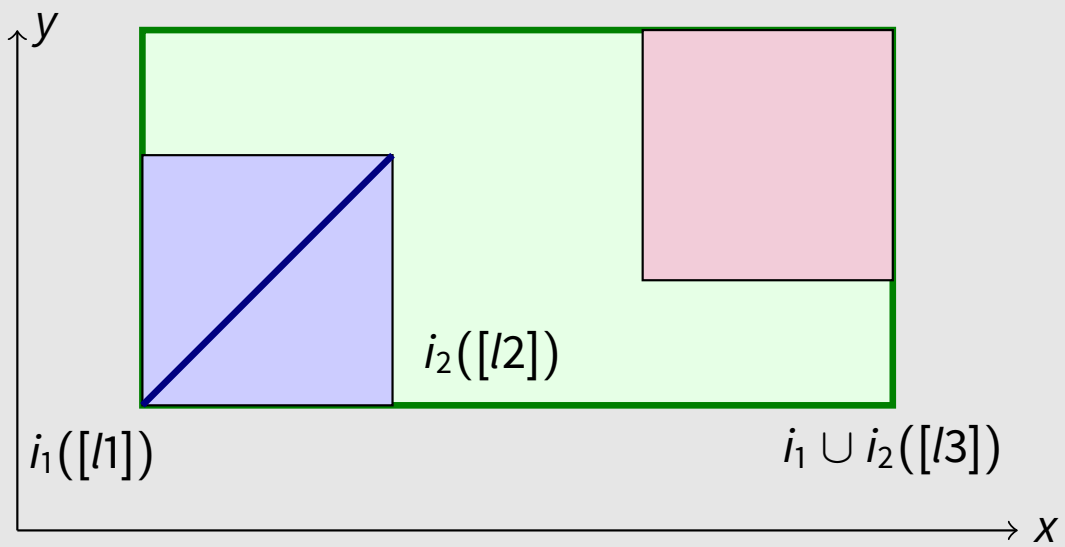
Abstraction and control flow joins

In case of conditional structures

- We must over-approximate the union of abstract values computed in the different branches
- We may lose some precision, but not forget behaviors

Example (Intervals)

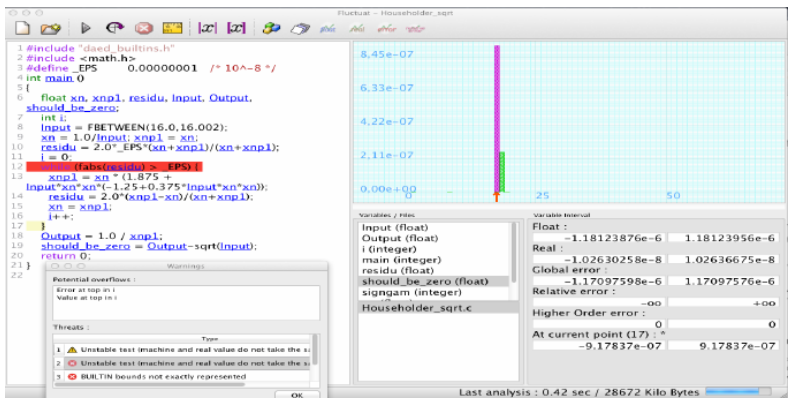
```
int x,y;
if (random(0,1)<0.5) {
  x=2+rand(0,2);
  y=x;  [l1]  }
else {   x=6+rand(0,2) ;
  y=3+rand(0,2); [l2]
  }
[l3]
```

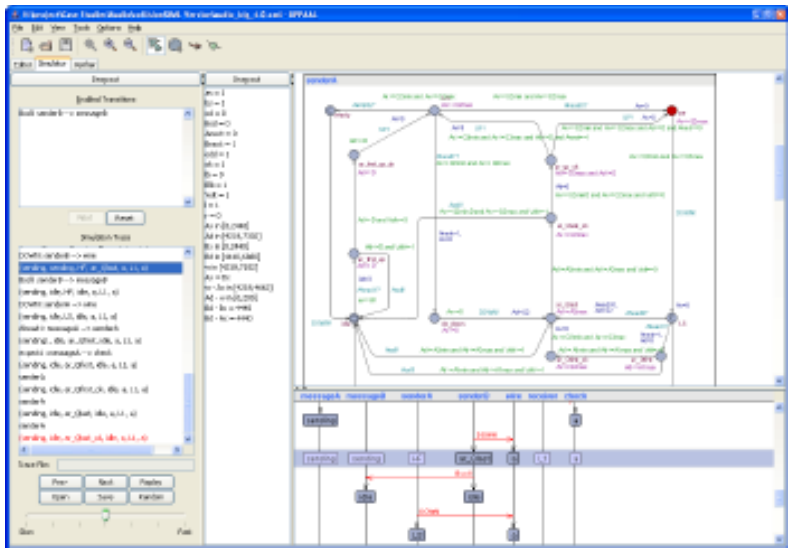


Non relational abstraction: we join abstract values componentwise on x and y

Program verification: classes of formal methods

- Abstract interpreters (Astree, Fluctuat etc.) : synthetize properties (in a limited set of "abstract" properties) automatically, directly on programs/"code" of systems
- Model-checkers (Uppaal etc.) : checking properties automatically on models of computation/systems; SAT/SMT solvers
- Interactive provers (Frama-C, Keymaera, COQ, etc.) : interactively prove/discover properties on code/system





Back to the Verification of neural networks

Is it different from classical program verification?

We could consider neural networks as a very specific class of programs and apply traditional program verification but ...

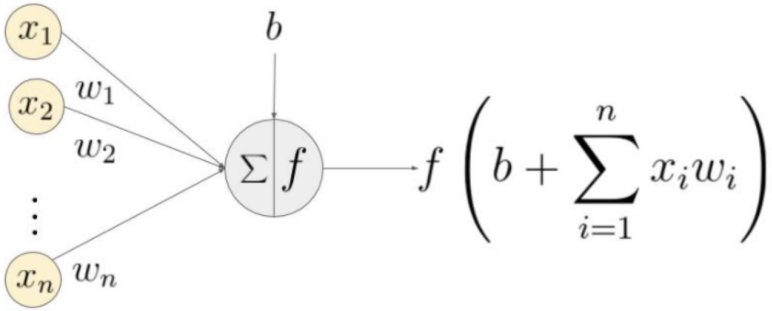
- Neural networks are often very large: new specific and scalable abstractions are needed (that exploit the structure of networks)
  - ReLU could be encoded as a `if then else` block, but that's not the best way
- Their internal workings are not perfectly understood:
  - they are learned from data instead of written by a human being
  - a specific weight or part of a network cannot be pointed out as the cause of a behavior
  - local/compositional reasoning almost impossible?
- Many applications lack specifications (for example, if the spec is that the network must recognize a stop sign if any, how do we mathematically specify?)

On the other hand, the simplest classes of neural networks such as feedforward neural networks have no loop, so fixed point computation, one of the most difficult problems in program analysis, is avoided (but useful for recurrent networks).

Verification of neural networks: activation functions

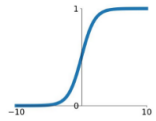
Relies on abstractions from program verification but customized for neural networks.

One neuron is

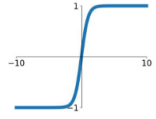


with nonlinear activation function f among

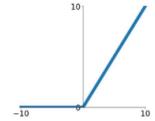
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



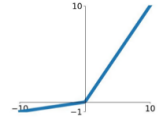
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

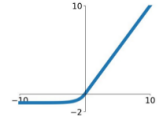


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



RELU activation function

- Very commonly used in applications, and the first target of verification approaches
- Piecewise linear activation function: easy encoding to linear arithmetic constraints

Verification of Feedforward Neural Networks with ReLU activation

Conceptually a function, but non-convexity makes verification NP-complete.
Each neuron is conceptually a switch (2^N configurations).

Affine forms and the Zonotope abstraction

Each variable x_k is represented by an Affine Form

$$\hat{x_1} = \alpha_0^1 + \sum_{i=1}^{p} \alpha_i^1 \epsilon_i$$

$$\dots$$

$$\hat{x_n} = \alpha_0^n + \sum_{i=1}^{p} \alpha_i^n \epsilon_i$$

- the ϵ_i are symbolic variables called noise symbols, which range is [-1,1]
- the α_i^k express the amplitude of the corresponding noise in the variable
- sharing ϵ_i between the variables $x_1, \dots, x_n$ implicitely expresses dependency

Affine forms and the Zonotope abstraction

Each variable x_k is represented by an Affine Form

$$\hat{x_1} = \alpha_0^1 + \sum_{i=1}^{p} \alpha_i^1 \epsilon_i$$

$$\dots$$

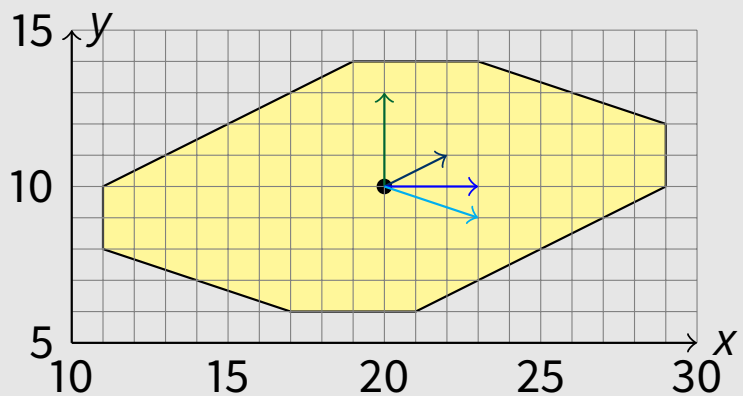
$$\hat{x_n} = \alpha_0^n + \sum_{i=1}^{p} \alpha_i^n \epsilon_i$$

- the ϵ_i are symbolic variables called noise symbols, which range is [-1,1]
- the α_i^k express the amplitude of the corresponding noise in the variable
- sharing ϵ_i between the variables $x_1, \dots, x_n$ implicitely expresses dependency

Geometric concretization as zonotopes in $\mathbb{R}^n$

- a center-symmetric polytope which faces are also center-symmetric
- the projection in $\mathbb{R}^n$ of a linear of a p-dimensional cube

$$\begin{aligned} \hat{x} &= 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4 \\ \hat{y} &= 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4 \end{aligned}$$



Zonotope Transformers

Affine transformers are exact

Uncertain initial value given in $[a, b]$ for an input variable x introduces a fresh noise symbol ε_i and creates a centered form:

$$\hat{x} = \frac{(a+b)}{2} + \frac{(b-a)}{2}\varepsilon_i$$

Multplication by a constant $\lambda \in \mathbb{R}$ and addition are computed component-wise and exact:

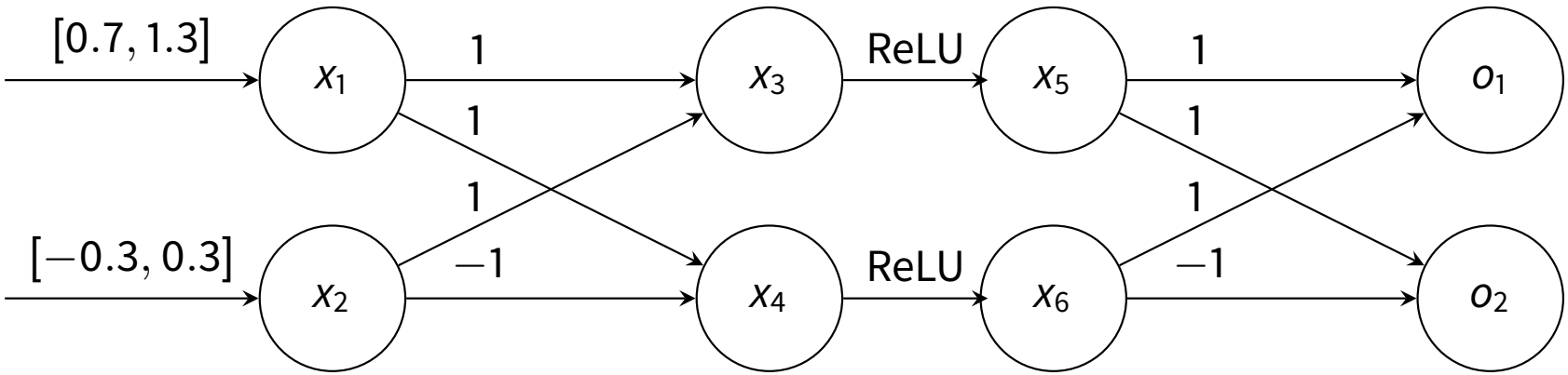
$$\lambda\hat{x} = \lambda x_0 + \lambda x_1\varepsilon_1 + \ldots + \lambda x_p\varepsilon_p$$
$$\hat{x} + \hat{y} = (x_0 + y_0) + (x_1 + y_1)\varepsilon_1 + \ldots + (x_p + y_p)\varepsilon_p$$

Custom ReLU transformer

The activation functions (e.g. ReLU) need to be abstracted, precision is lost in general.

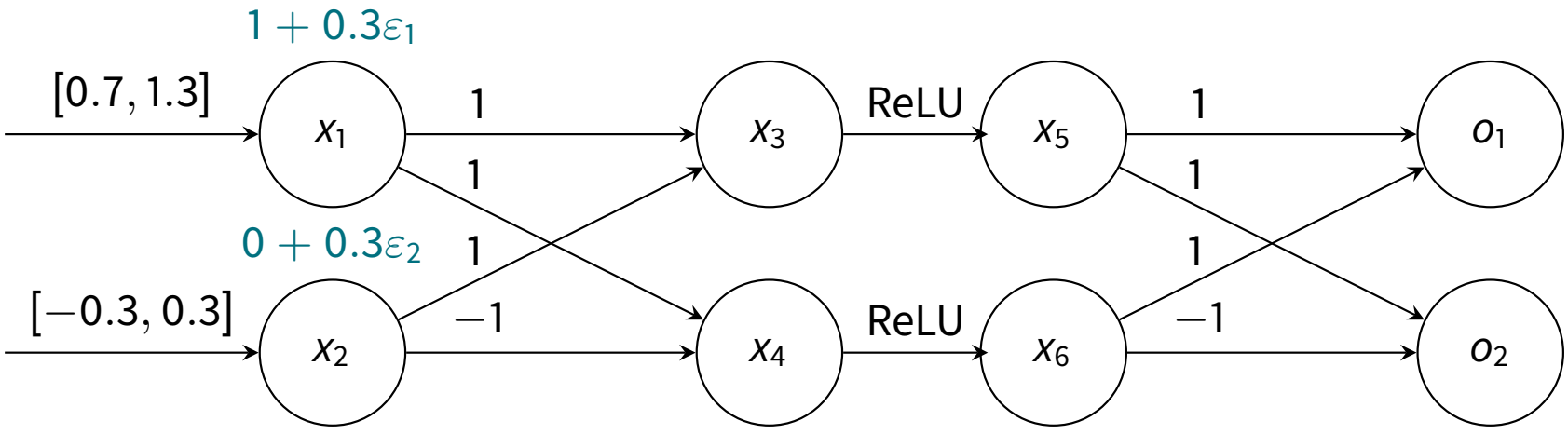
Coming back to the example where Boxes fail to prove robustness

Exercise: do zonotopes succeed in verifying robustness ?



Coming back to the example where Boxes fail to prove robustness

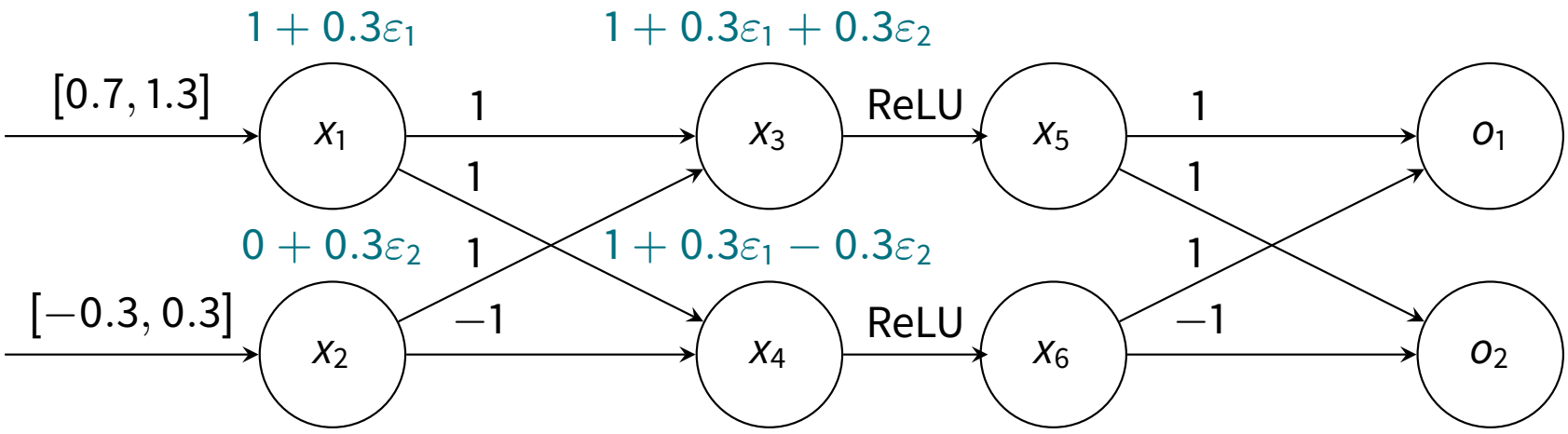
Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$

Coming back to the example where Boxes fail to prove robustness

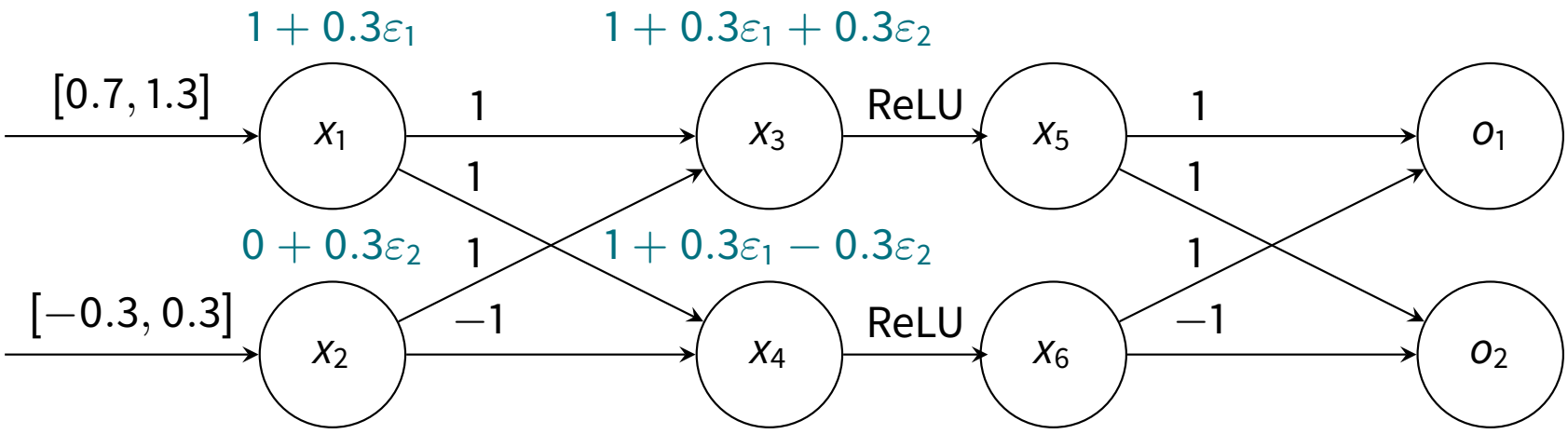
Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$

Coming back to the example where Boxes fail to prove robustness

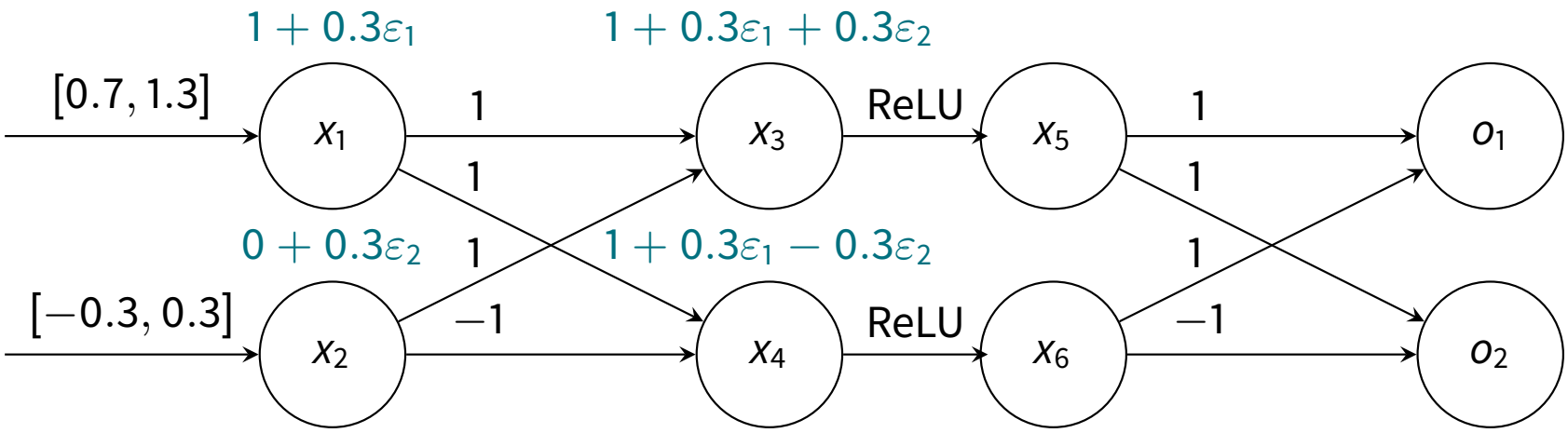
Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$
- can you draw the zonotope for (x_3, x_4)?

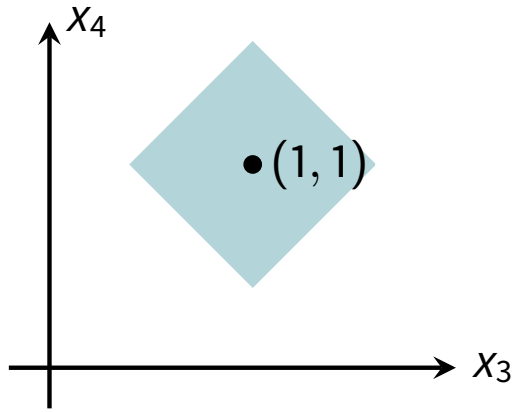
Coming back to the example where Boxes fail to prove robustness

Exercise: do zonotopes succeed in verifying robustness ?



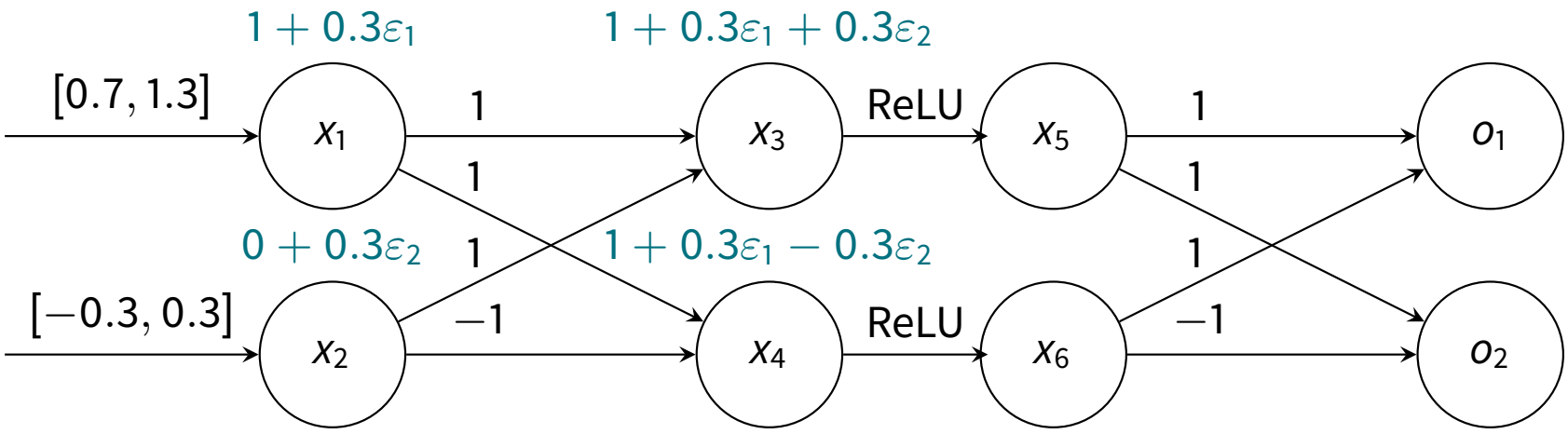
- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$
- can you draw the zonotope for (x_3, x_4)?

$$\begin{pmatrix} \hat{x}_3 \\ \hat{x}_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.3 \\ 0.3 \end{pmatrix} \varepsilon_1 + \begin{pmatrix} 0.3 \\ -0.3 \end{pmatrix} \varepsilon_2$$



Coming back to the example where Boxes fail to prove robustness

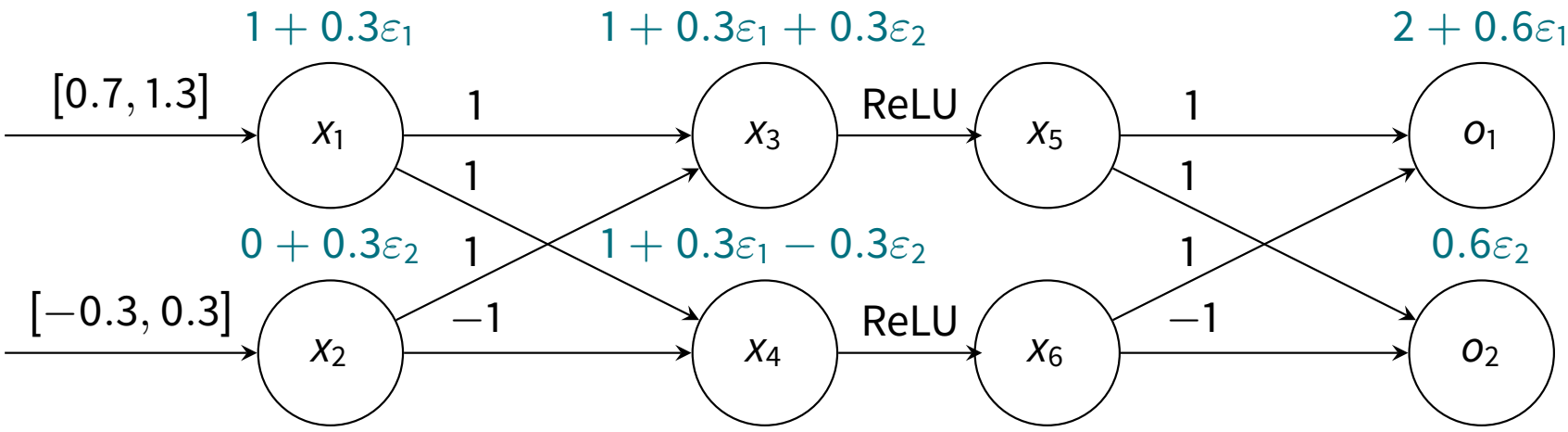
Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$
- can you draw the zonotope for (x_3, x_4)?
- ReLU layer: $\hat{x}_5 = \max(\hat{x}_3, 0) = \hat{x}_3$ and $\hat{x}_6 = \max(\hat{x}_4, 0) = \hat{x}_4$ because $\hat{x}_3 \in [0.4, 1.6] \geq 0$ and $\hat{x}_4 \in [0.4, 1.6] \geq 0$ for $\varepsilon_1, \varepsilon_2 \in [-1, 1]$.

Coming back to the example where Boxes fail to prove robustness

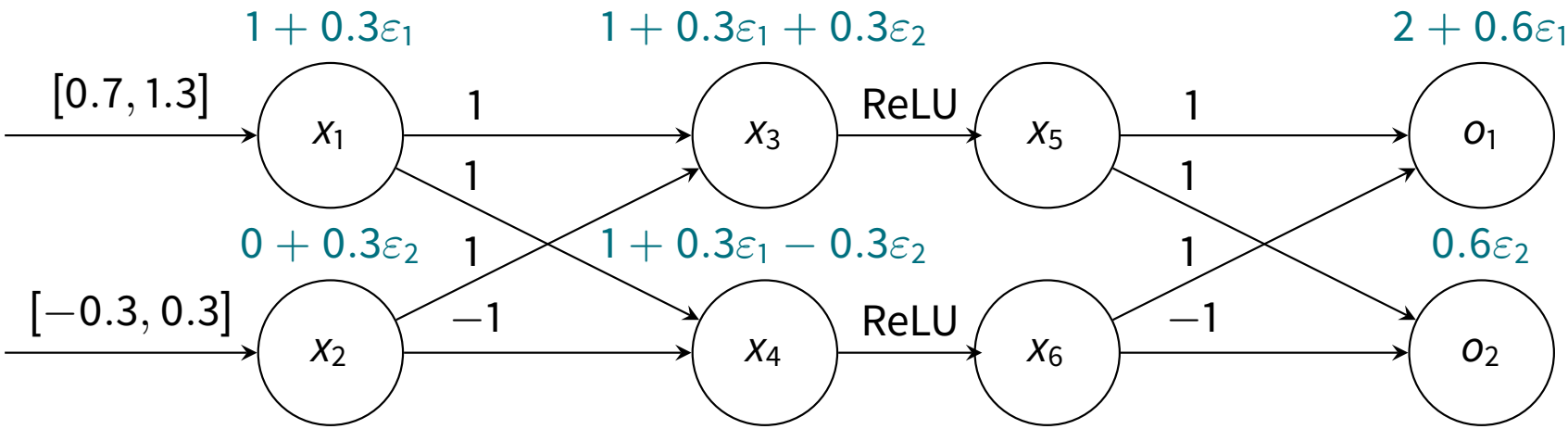
Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$
- can you draw the zonotope for (x_3, x_4)?
- ReLU layer: $\hat{x}_5 = \max(\hat{x}_3, 0) = \hat{x}_3$ and $\hat{x}_6 = \max(\hat{x}_4, 0) = \hat{x}_4$ because $\hat{x}_3 \in [0.4, 1.6] \geq 0$ and $\hat{x}_4 \in [0.4, 1.6] \geq 0$ for $\varepsilon_1, \varepsilon_2 \in [-1, 1]$.
- linear transforms: $o_1 = x_5 + x_6 = 2 + 0.6\varepsilon_1$ and $o_2 = x_5 - x_6 = 0.6\varepsilon_2$

Coming back to the example where Boxes fail to prove robustness

Exercise: do zonotopes succeed in verifying robustness ?



- inputs: fresh noise symbols and centered forms $\hat{x}_1 = 1 + 0.3\varepsilon_1$ and $\hat{x}_2 = 0 + 0.3\varepsilon_2$
- linear transforms: $\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.3\varepsilon_1 + 0.3\varepsilon_2, \hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 1 + 0.3\varepsilon_1 - 0.3\varepsilon_2$
- can you draw the zonotope for (x_3, x_4)?
- ReLU layer: $\hat{x}_5 = \max(\hat{x}_3, 0) = \hat{x}_3$ and $\hat{x}_6 = \max(\hat{x}_4, 0) = \hat{x}_4$ because $\hat{x}_3 \in [0.4, 1.6] \geq 0$ and $\hat{x}_4 \in [0.4, 1.6] \geq 0$ for $\varepsilon_1, \varepsilon_2 \in [-1, 1]$.
- linear transforms: $o_1 = x_5 + x_6 = 2 + 0.6\varepsilon_1$ and $o_2 = x_5 - x_6 = 0.6\varepsilon_2$
- success in deducing $o_1 \geq o_2$ because $o_1 \in [1.4, 2.6]$ and $o_2 \in [-0.6, 0.6]$

Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

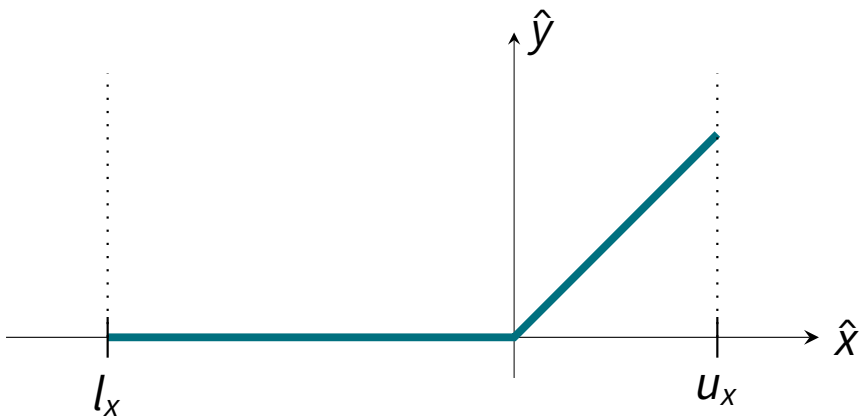
For

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_p\varepsilon_p,\ \ \varepsilon_1, \ldots \varepsilon_p \in [-1, 1]$$

we can bound the values reachable by x by

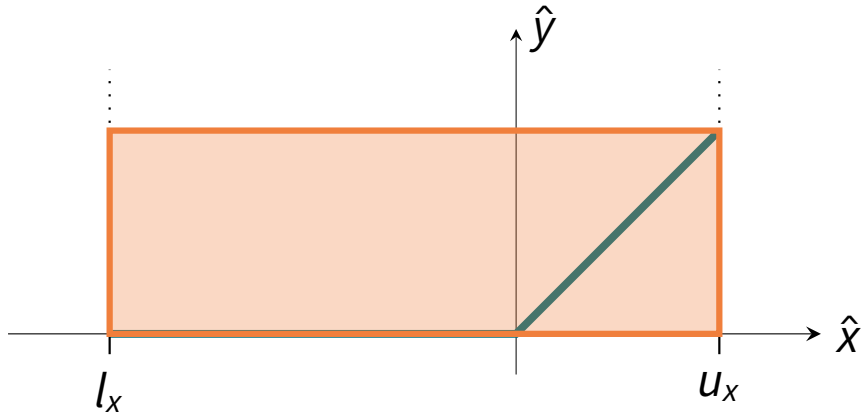
$$[l_x, u_x] = [x_0 - \sum_{i=1}^{p} |x_i|, x_0 + \sum_{i=1}^{p} |x_i|]$$

- if $l_x \geq 0$ then $\hat{y} = \hat{x}$
- if $u_x \leq 0$ then $\hat{y} = 0$
- otherwise ?



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

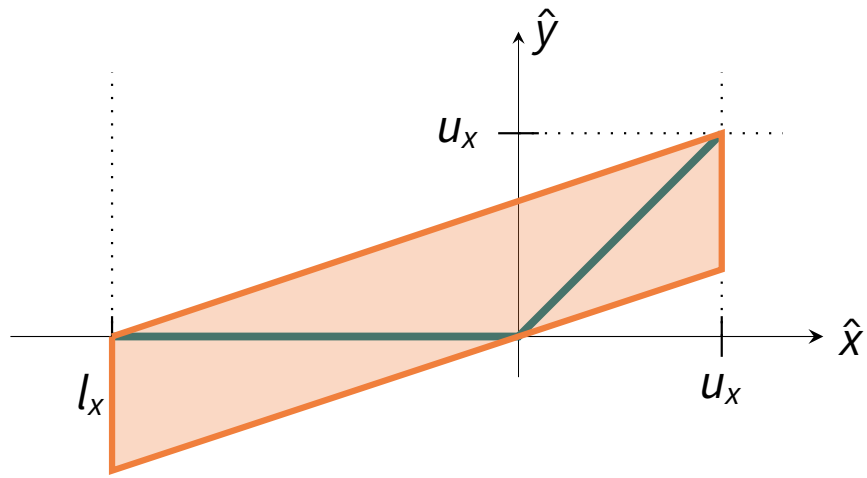
First option: a Box



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: find a Zonotope enclosing the ReLU.

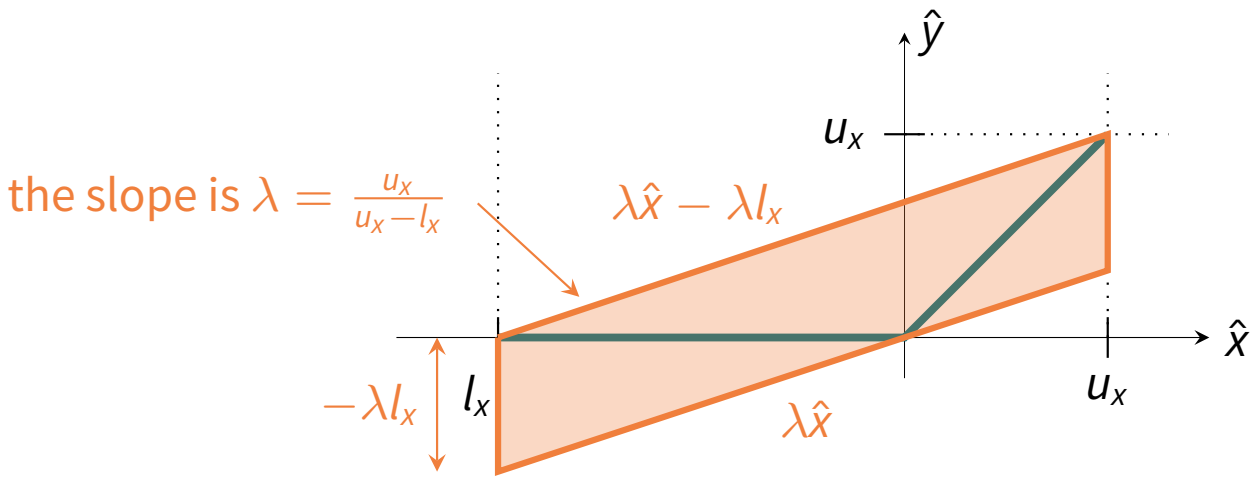
- define two parallel lines (otherwise we don't have a zonotope)



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: find a Zonotope enclosing the ReLU.

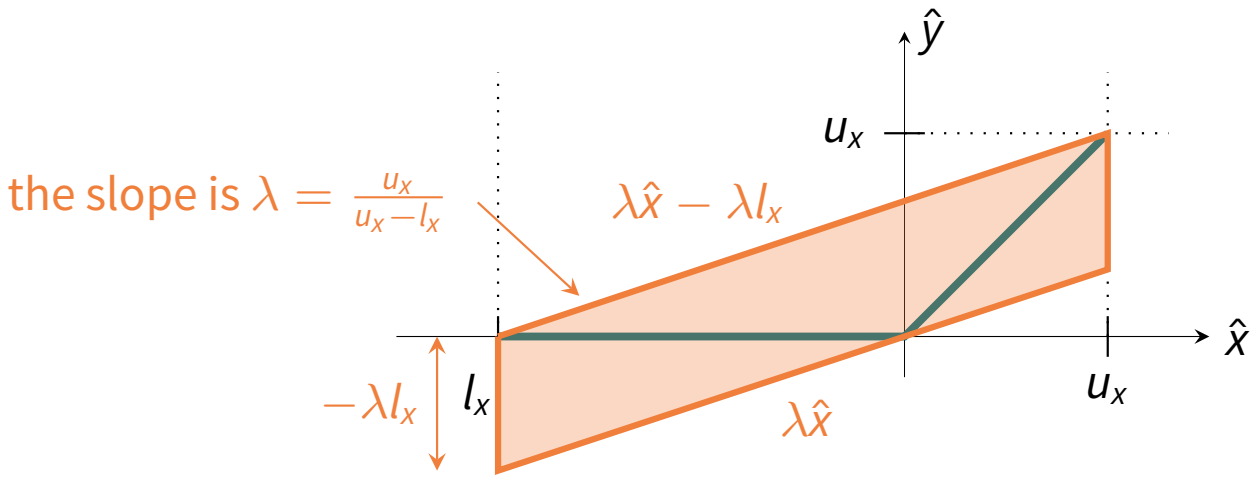
- define two parallel lines (otherwise we don't have a zonotope)
- parameterized by $\lambda = \frac{u_x}{u_x - l_x}$: lower line is $\lambda\hat{x}$, upper line is $\lambda\hat{x} - \lambda l_x$



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: find a Zonotope enclosing the ReLU.

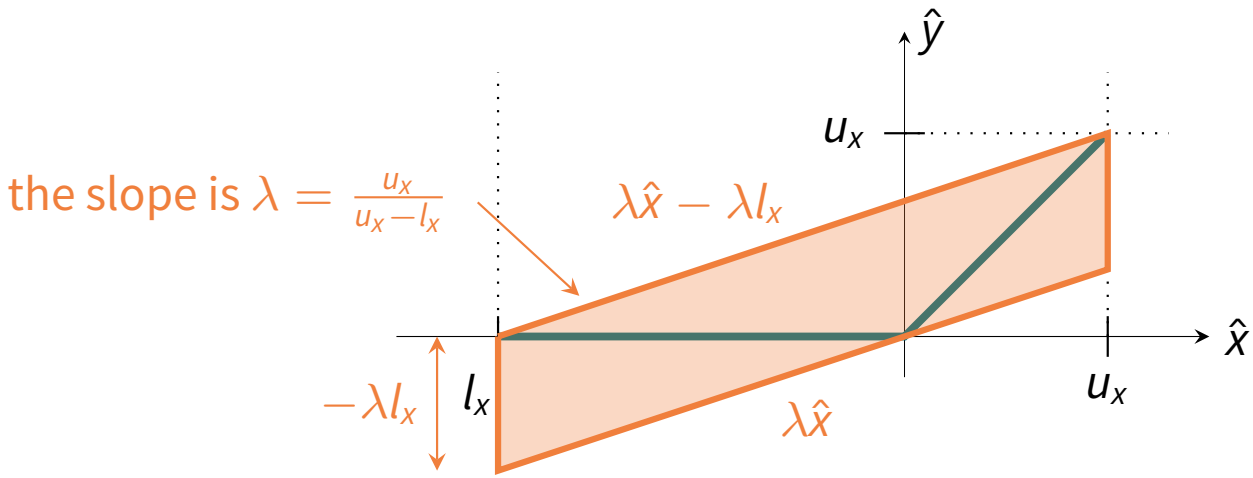
- define two parallel lines (otherwise we don't have a zonotope)
- parameterized by $\lambda = \frac{u_x}{u_x - l_x}$: lower line is $\lambda\hat{x}$, upper line is $\lambda\hat{x} - \lambda l_x$
- from $\lambda\hat{x} \leq \hat{y} \leq \lambda\hat{x} - \lambda l_x$, deduce $\hat{y} = \lambda\hat{x} - \frac{\lambda l_x}{2} - \frac{\lambda l_x}{2}\varepsilon_{new}$



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: find a Zonotope enclosing the ReLU.

- define two parallel lines (otherwise we don't have a zonotope)
- parameterized by $\lambda = \frac{u_x}{u_x - l_x}$: lower line is $\lambda\hat{x}$, upper line is $\lambda\hat{x} - \lambda l_x$
- from $\lambda\hat{x} \leq \hat{y} \leq \lambda\hat{x} - \lambda l_x$, deduce $\hat{y} = \lambda\hat{x} - \frac{\lambda l_x}{2} - \frac{\lambda l_x}{2}\varepsilon_{new}$



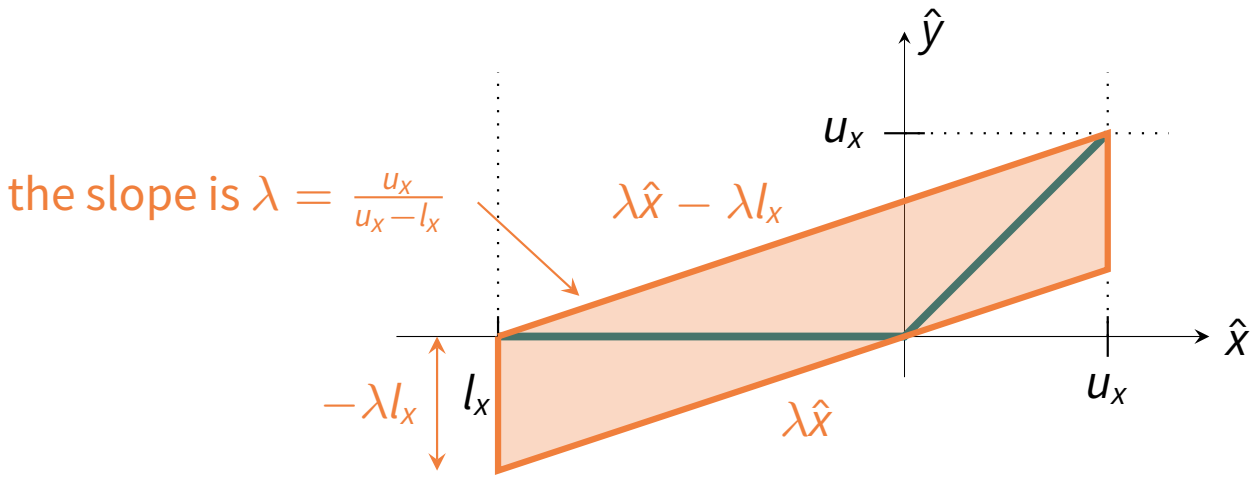
Example (exercise):

$\hat{x} = -0.5 + 1.5\varepsilon_1,$

Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: find a Zonotope enclosing the ReLU.

- define two parallel lines (otherwise we don't have a zonotope)
- parameterized by $\lambda = \frac{u_x}{u_x - l_x}$: lower line is $\lambda\hat{x}$, upper line is $\lambda\hat{x} - \lambda l_x$
- from $\lambda\hat{x} \leq \hat{y} \leq \lambda\hat{x} - \lambda l_x$, deduce $\hat{y} = \lambda\hat{x} - \frac{\lambda l_x}{2} - \frac{\lambda l_x}{2}\varepsilon_{new}$



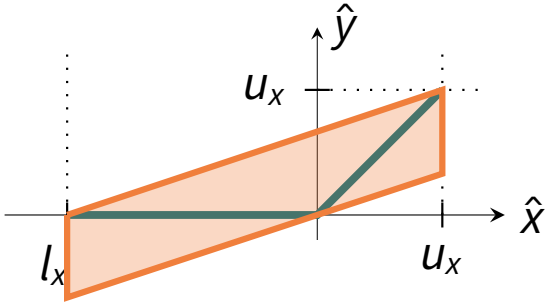
Example (exercise):

$\hat{x} = -0.5 + 1.5\varepsilon_1, \ [l_x, u_x] = [-2, 1]$

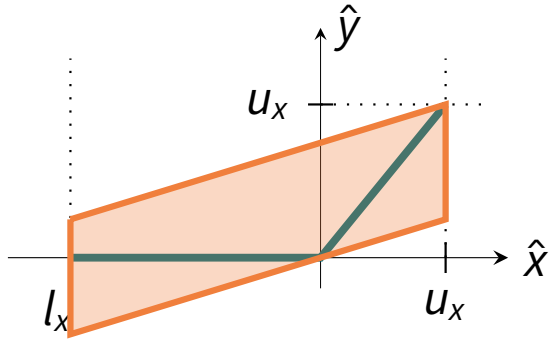
$$\lambda = \frac{u_x}{u_x - l_x} = \frac{1}{3} \implies \frac{1}{3}\hat{x} \leq \hat{y} \leq \frac{1}{3}\hat{x} + \frac{2}{3} \implies \hat{y} = \frac{1}{3}\hat{x} + \frac{1}{3} + \frac{1}{3}\varepsilon_2 = \frac{1}{6} + \frac{1}{2}\varepsilon_1 + \frac{1}{3}\varepsilon 2$$

Other Zonotopes transformers for RELU are possible

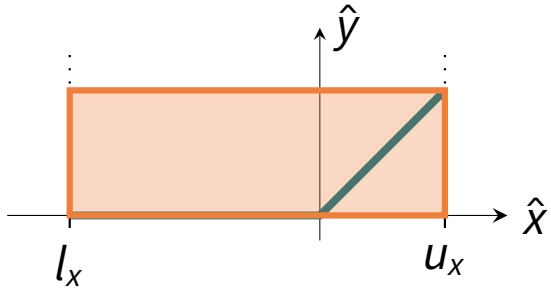
- There are many non comparable Zonotope transformers: not one zonotope is smaller in terms of included in the others
- Even the Box transformer is not strictly comparable
- The one of the previous slide is minimal in term of area in the input-output plane



$$\lambda = \frac{u_x}{u_x - l_x} = \frac{1}{3}$$



$$\lambda = \frac{1}{4}$$



$$\lambda = 0$$

References:

- AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, IEEE S&P 2018
- Fast and Effective Robustness Certification, G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev, NIPS 2018.

Lab session

Part 2: Simple abstractions for the reachability analysis of neural networks

- Tutorial on Lazy Sets
- Implementing a basic interval then zonotopic reachability analysis of simple ReLU feedforward networks