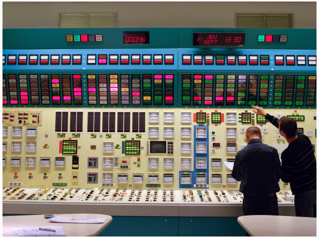
Formal Verification of Machine Learning Techniques

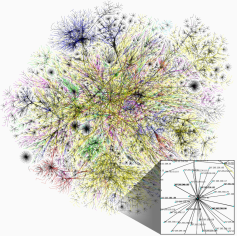
Julien Girard-Satabin (CEA LIST/INRIA TAU)

Pawan Kumar, rapporteur (Oxford University)
Antoine Miné, rapporteur (Université Paris-Sorbonne)
Sylvie Putot, examinatrice (LIX)
Gilles Dowek, examinateur (INRIA)
Caterina Urban, examinatrice (INRIA)
Guillaume Charpiat, co-encadrant (INRIA)
Zakaria Chihani, co-encadrant (CEA LIST)
Marc Schoenauer, directeur (INRIA)

February 15, 2022

On trusting programs





Software is interlinked with human activities

On the necessity to trust programs

Trust:

- Software needs to work
- Social acceptance for fair societies
- But trust is a complex notion...

On the necessity to trust programs

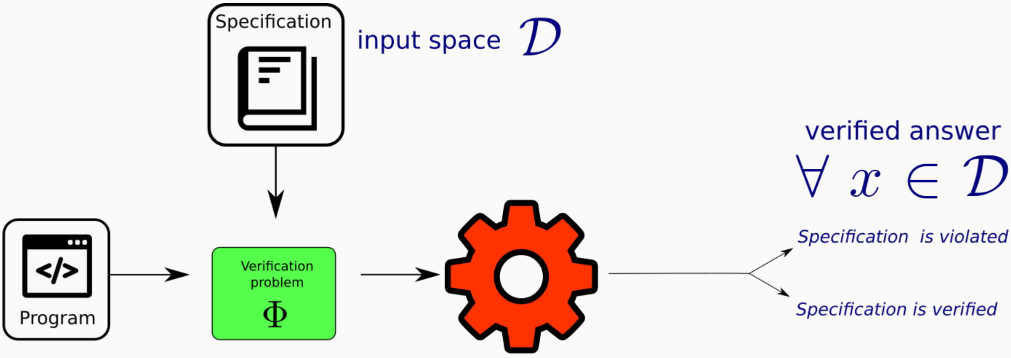
Trust:

- Software needs to work
- Social acceptance for fair societies
- But trust is a complex notion...

Reliability:

- Behaving consistently
- Regarding specified operating conditions

On the necessity to formally verify programs



Specification

input space $\mathcal{D}$

Program

Verification problem Φ

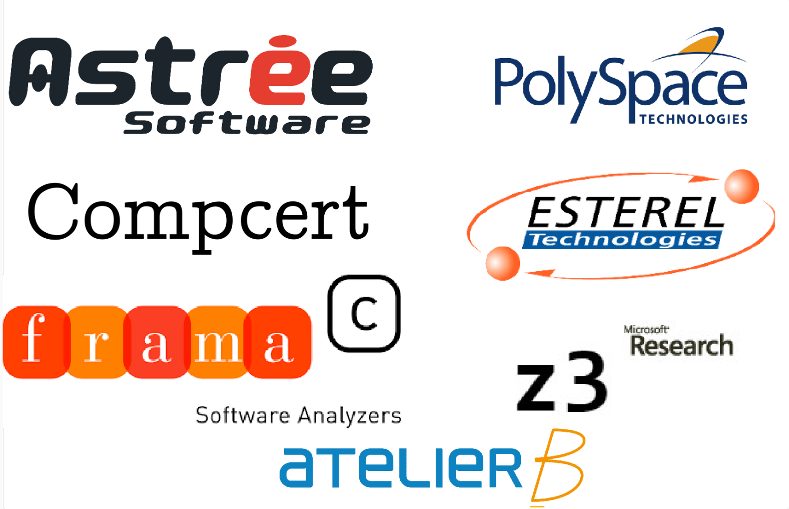
verified answer $\forall x \in \mathcal{D}$

Specification is violated

Specification is verified

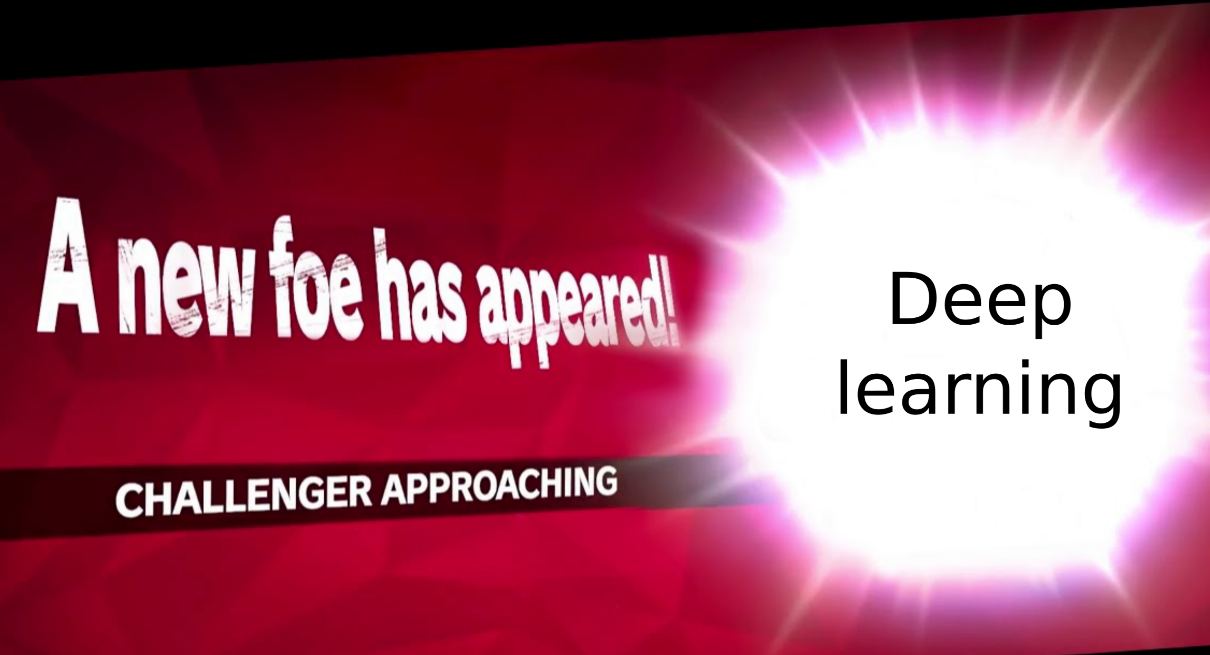
Formal verification

Formal verification is a success...



A new foe has appeared!

CHALLENGER APPROACHING



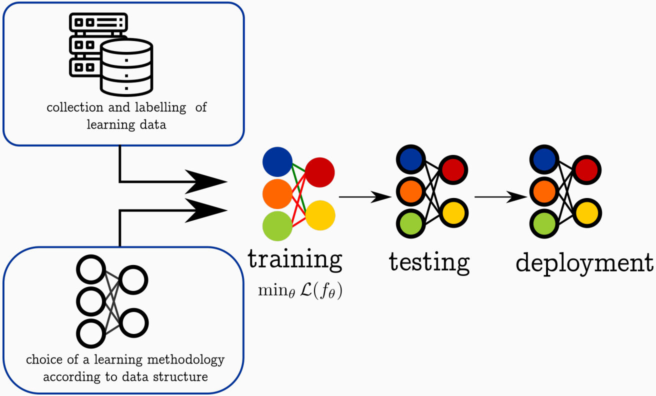
Deep learning

What deep learning programming is

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

All credits to Randall Munroe

What deep learning programming is

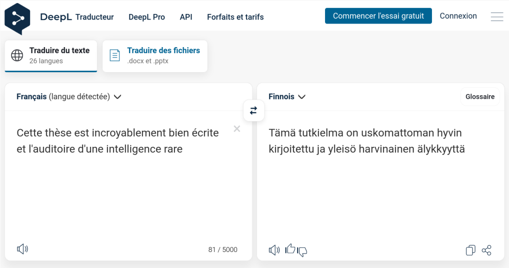

collection and labelling of learning data
choice of a learning methodology according to data structure
training
$\min_\theta \mathcal{L}(f_\theta)$
testing
deployment

cat 99%

horse 98%

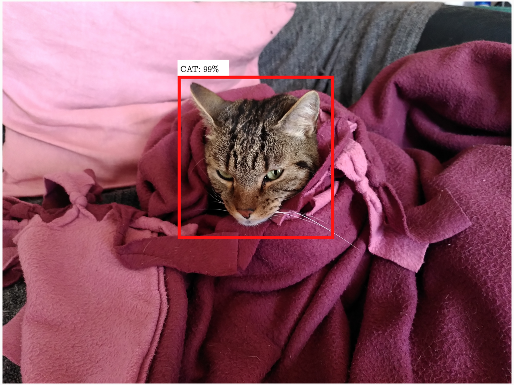
- software that takes data and performance criterion as specification (for instance: loss function)
- training modifies the base program until sufficient performance levels are reached

What deep learning programming allows



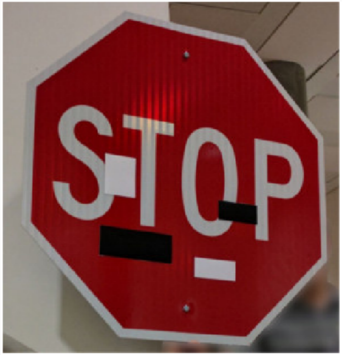
Cette thèse est incroyablement bien écrite et l'auditoire d'une intelligence rare

Tämä tutkielma on uskomattoman hyvin kirjoitettu ja yleisö harvinainen älykkyyttä



Natural language processing, object detection. . . pattern detection on ***perceptive inputs*** (inputs we perceive as humans) of high dimension ($400 \times 300 \times 3 = 360000$ values to describe Ernest)

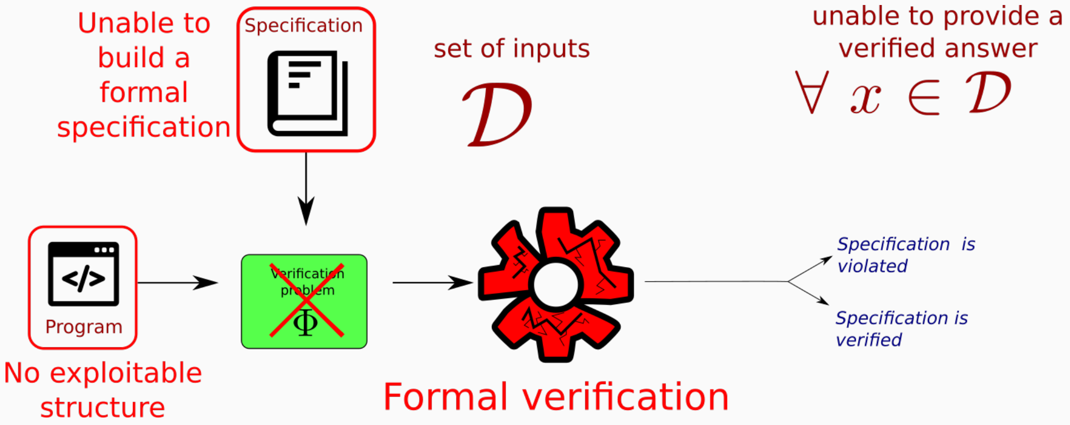
How deep learning programs (may) fail



from Robust Physical-World Attacks on Deep Learning Visual Classification, Eykholt, Evtimov et al., CVPR 2018

NTSB investigations on Uber and Tesla

What deep learning broke in the verification process



Unable to build a formal specification

Specification

set of inputs

$\mathcal{D}$

unable to provide a verified answer

$\forall x \in \mathcal{D}$

Program

Verification problem

Φ

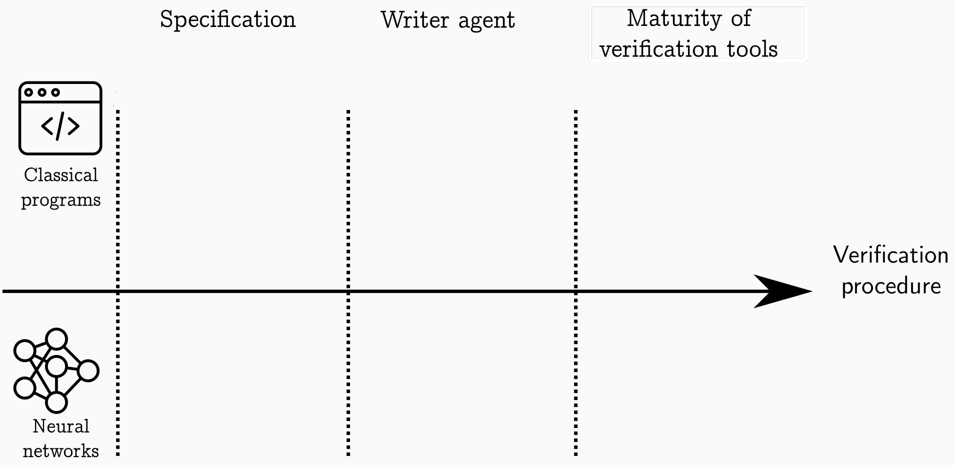
Specification is violated

Specification is verified

No exploitable structure

Formal verification

What is at stake?



Specification

Writer agent

Maturity of verification tools

Classical programs

Neural networks

Verification procedure

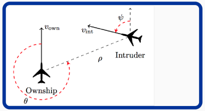
What is at stake?

Specification

Writer agent

Maturity of verification tools

Classical programs



low dimensional input, explicit specification

Verification procedure

Neural networks

high dimensional input, data as partial specification

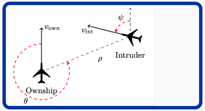
What is at stake?

Specification

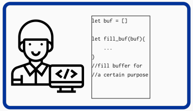
Writer agent

Maturity of verification tools

Classical programs



low dimensional input, explicit specification



```
let buf = []

let fill_buf(buf){
   ...
}
//fill buffer for
//a certain purpose
```

a programmer/ automaton directly writes the program

Verification procedure

Neural networks

high dimensional input, data as partial specification

$$\min_\theta \mathcal{L}(\theta)$$

a function minimization indirectly writes the program

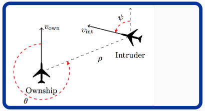
What is at stake?

Specification

Writer agent

Maturity of verification tools

Classical programs



low dimensional input, explicit specification

```
let buf = []

let fill_buf(buf){
    ...
}
//fill buffer for
//a certain purpose
```

a programmer/ automaton directly writes the program

Astrée Software
PolySpace Technologies
Compcert
ESTEREL Technologies
frama C
z3 Microsoft Research
Software Analyzers ateLier B

mature tools, multiple properties can be written, efficient heuristics

Verification procedure

Neural networks

high dimensional input, data as partial specification

$$\min_\theta \mathcal{L}(\theta)$$

a function minimization indirectly writes the program

still experimental, less expressive combinatorial explosion

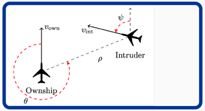
What is at stake?

Specification

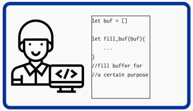
Writer agent

Maturity of verification tools

Classical programs



low dimensional input, explicit specification



```
let buf = []

let fill_buf(buf){
    ...
}
//fill buffer for
//a certain purpose
```

a programmer/ automaton directly writes the program

mature tools, multiple properties can be written, efficient heuristics

☺ ✓

Verification procedure

Neural networks

high dimensional input, data as partial specification

$$\min_\theta \mathcal{L}(\theta)$$

a function minimization indirectly writes the program

? ? ?

still experimental, less expressive combinatorial explosion

☹ ✗

What is at stake?

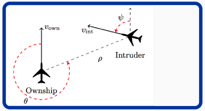
This Thesis

Specification

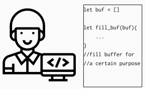
Writer agent

Maturity of verification tools

Classical programs



low dimensional input, explicit specification



```
let buf = []

let fill_buf(buf){
    ...
}
//fill buffer for
//a certain purpose
```

a programmer/ automaton directly writes the program

mature tools, multiple properties can be written, efficient heuristics

☺ ✓

Verification procedure

Neural networks

high dimensional input, data as partial specification

$$\min_\theta \mathcal{L}(\theta)$$

a function minimization indirectly writes the program

? ? ?

still experimental, less expressive combinatorial explosion

☹ ✗

Specification

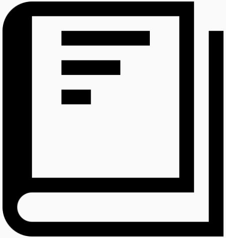
How to write proper **specifications** for deep learning software?

Tooling

How to improve the machinery of traditional solvers to **scale** on deep learning software?

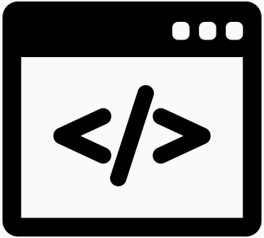
The specification problem

What do we need to formalize?



a specification

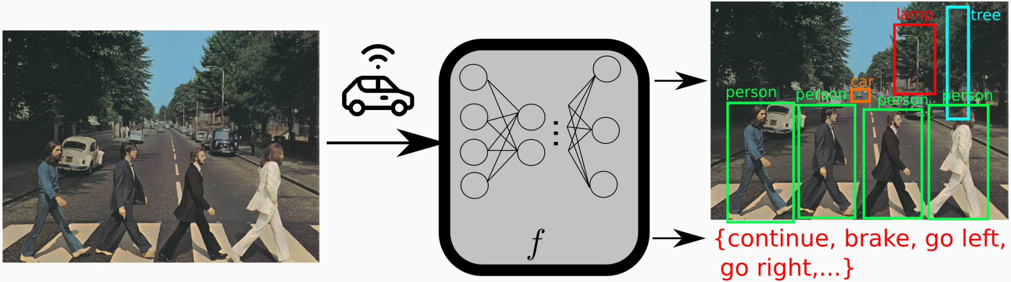
no formally specifiable inputs



a program

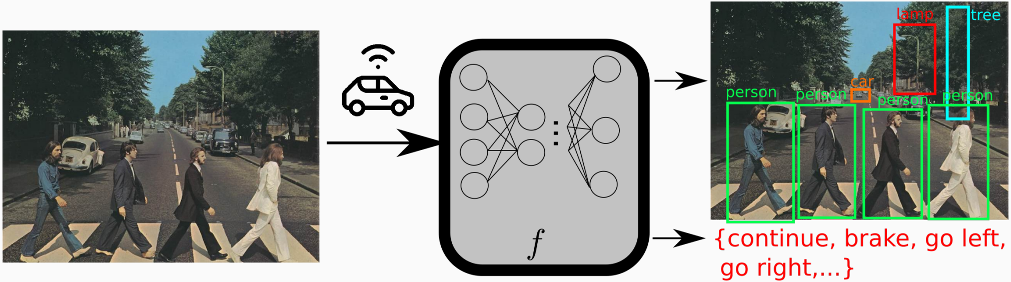
no exploitable structure

Running example: perception unit



{continue, brake, go left, go right,...}

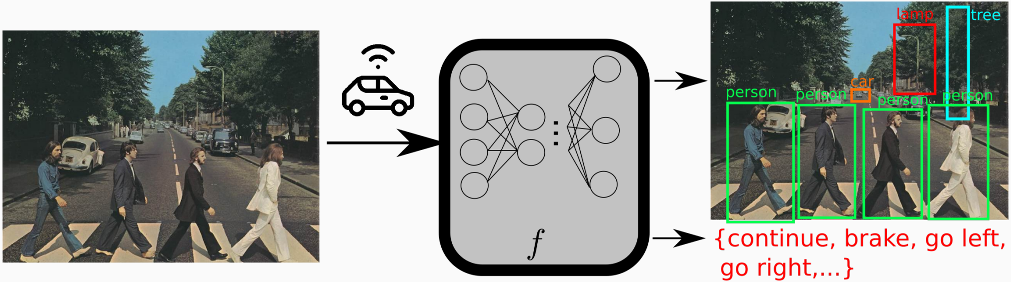
Running example: perception unit



{continue, brake, go left, go right,...}

Dream property: *for all images that contain a pedestrian, the autonomous car will never take a decision that would result in running over perceived pedestrians*

Running example: perception unit

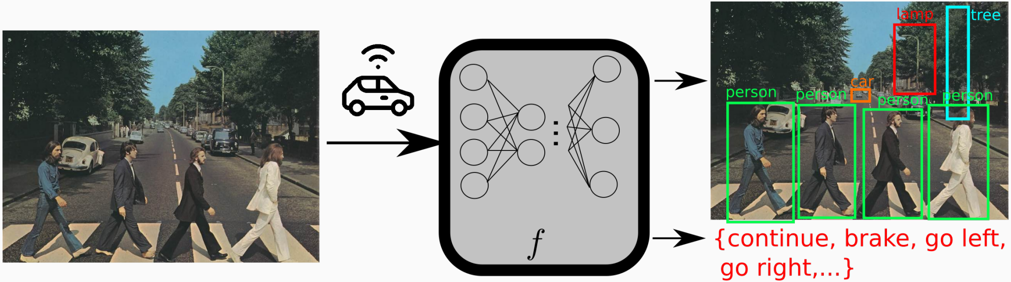


{continue, brake, go left, go right,...}

Dream property: *for all images that contain a pedestrian, the autonomous car will never take a decision that would result in running over perceived pedestrians*

no formal characterization of what an image with a pedestrian is!

Running example: perception unit



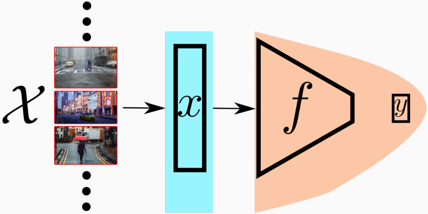
{continue, brake, go left, go right,...}

Dream property: *for all images that contain a pedestrian, the autonomous car will never take a decision that would result in running over perceived pedestrians*

no formal characterization of what an image with a pedestrian is!

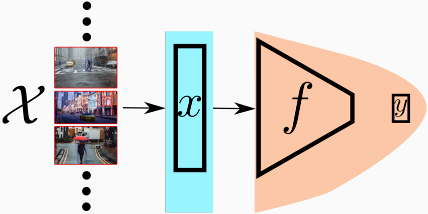
Lack of formal definition on inputs $\implies$ ***no relevant safety properties***

Formalization



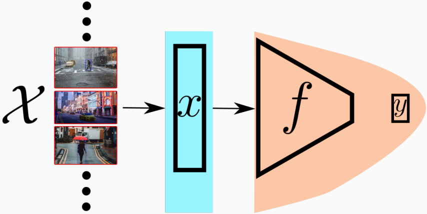
- $\mathcal{X}$: input space
- $x \in \mathcal{X}$: input sample
- f: decision function
- y: output

Formalization



- $\mathcal{X}$: input space: *no formal definition*
- $x \in \mathcal{X}$: input sample
- f: decision function: *no exploitable structure*
- y: output: *fixed format*, but unknown value for data outside of the training set

Formalization

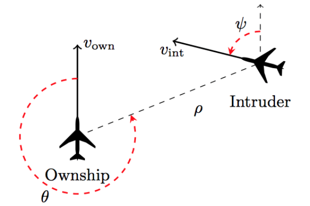


- $\mathcal{X}$: input space: *no formal definition*
- $x \in \mathcal{X}$: input sample
- f: decision function: *no exploitable structure*
- y: output: *fixed format*, but unknown value for data outside of the training set

no property to verify, thus ***no formal specification***

Special cases where formal verification is possible

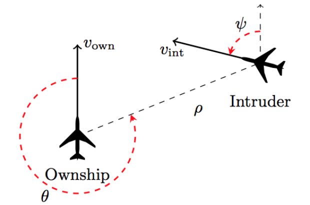
replacing programs with an existing semantic (*e.g.*, ACAS-Xu)



Global properties on existing semantic

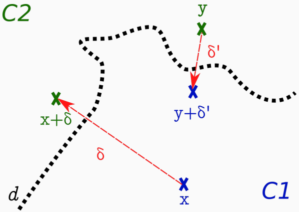
Special cases where formal verification is possible

replacing programs with an existing semantic (*e.g.*, ACAS-Xu)



Global properties on existing semantic

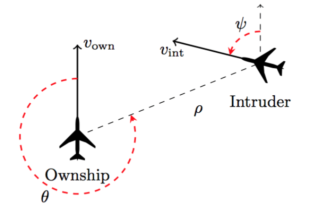
working on local perceptual inputs (*e.g.*, adversarial robustness)



Local properties on perceptual inputs

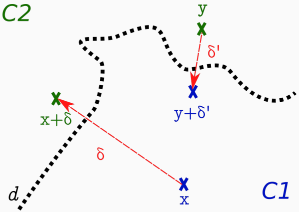
Special cases where formal verification is possible

replacing programs with an existing semantic (*e.g.*, ACAS-Xu)



Global properties on existing semantic

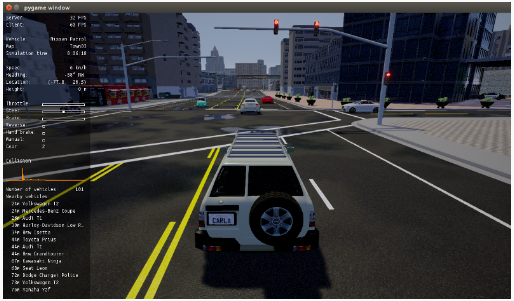
working on local perceptual inputs (*e.g.*, adversarial robustness)



Local properties on perceptual inputs

We aim to provide ***global properties on perceptual inputs***

Simulators in the industry



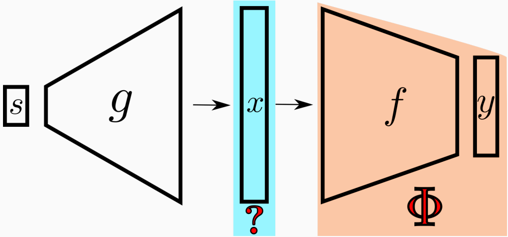
```
Server:                    32 FPS
Client:                    60 FPS

Vehicle:            Nissan Patrol
Map:                       Town03
Simulation time:          0:06:10

Speed:                     6 km/h
Heading:                  -88° NW
Location:           (-77.8,  28.5)
Height:                       0 m

Throttle:
Steer:
Brake:
Reverse:    □
Hand brake: □
Manual:     □
Gear:       2

Collision

Number of vehicles:        101
Nearby vehicles:
  24m Volkswagen T2
  24m Mercedes-Benz Coupe
  26m Audi Tt
  30m Harley-Davidson Low R.
  34m Bmw Isetta
  44m Toyota Prius
  44m Audi Tt
  44m Bmw Grandtourer
  67m Kawasaki Ninja
  68m Seat Leon
  72m Dodge Charger Police
  73m Volkswagen T2
  76m Yamaha Yzf
```

Screenshot from the CARLA open source simulator

Pros of using simulators for deep learning programming:

- lower costs
- more control on the design
- better edge cases scenarios handling

Simulators as data providers

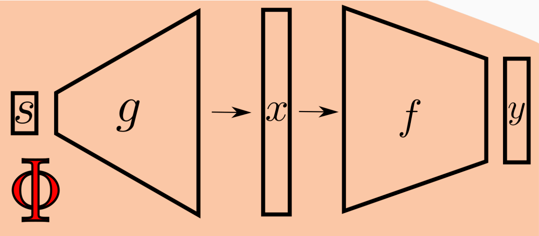


- $s \in \mathcal{S}$: scenario parameters (weather condition, location of pedestrian…)
- g: simulator
- x: perceptual input (images)

- f: model
- y: decision output (brake…)
- Φ: "$\forall x$ that contains a pedestrian, do not run over it"

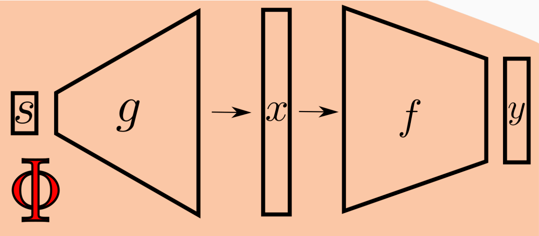
How to formulate Φ? What is an image x with a pedestrian?

Our approach



Modify the verification problem formulation to include g and s

Our approach

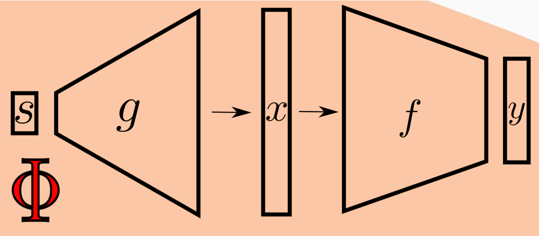


Modify the verification problem formulation to include g and s

Since s is part of Φ, Φ can now be expressed formally:

$$\forall s \in \mathcal{S} \text{ such that } \left\{ s_{pedestrian} \geq 1 \right\}, f(g(s)) = y_{brake}$$

Our approach



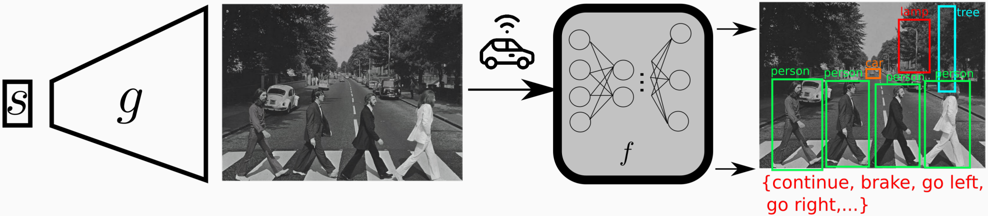
Modify the verification problem formulation to include g and s

Since s is part of Φ, Φ can now be expressed formally:

$$\forall s \in \mathcal{S} \text{ such that } \left\{ s_{pedestrian} \geq 1 \right\}, f(g(s)) = y_{brake}$$

We now have a property to verify a perceptive unit!

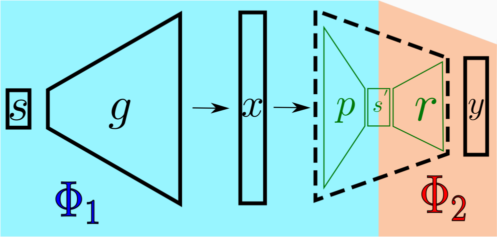
Our approach



Certifying Autonomous Models Using Simulators (CAMUS)[1]: put the burden of trust on the simulator's input space to achieve a specifiable set of inputs

[1] *CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators*, Girard-Satabin et al., ECAI 2020

Refinement: splitting perception and reasoning

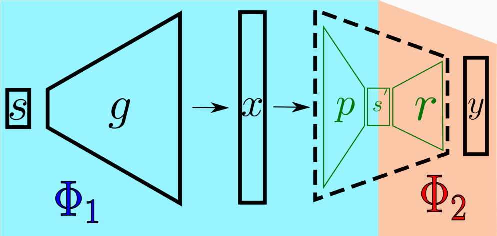


f splits in *p*erception and *r*easoning

Φ_1 on p: guarantee of no relevant information loss: *reconstruct s from x*
$\forall s, s = s^{'}$, which is phrased as $p \circ g(s) = s$

Φ_2 on r: do not kill pedestrians (assuming perfect perception), which is phrased as
$\forall s^{'}, \left\{ s^{'}_{pedestrian} \geq 1 \right\}, y = y_{brake}$

Refinement: splitting perception and reasoning



f splits in *p*erception and *r*easoning

Φ_1 on p: guarantee of controlled relevant information loss: *reconstruct s from x*
$\forall s, s \simeq s'$, which is phrased as $||p \circ g(s) - s|| < \varepsilon$

Φ_2 on r: do not kill pedestrians (assuming perfect perception), which is phrased as
$\forall s', \left\{ s'_{pedestrian} \geq 1 \right\}, y = y_{brake}$

How to implement CAMUS?

How to express Φ, g, f, $\mathcal{X}$?

How to implement CAMUS?

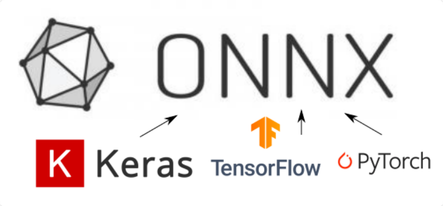
How to express Φ, g, f, $\mathcal{X}$?

At the beginning of this thesis (2017), there were less than five papers on formal verification of DNN (in 2021, several workshops, a competition...)

How to implement CAMUS?

How to express Φ, g, f, $\mathcal{X}$?

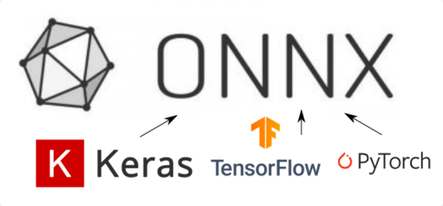
At the beginning of this thesis (2017), there were less than five papers on formal verification of DNN (in 2021, several workshops, a competition...)



How to implement CAMUS?

How to express Φ, g, f, $\mathcal{X}$?

At the beginning of this thesis (2017), there were less than five papers on formal verification of DNN (in 2021, several workshops, a competition...)



Bridging two existing standards to create an Inter Standard AI Encoding Hub (ISAIEH)

ISAIEH

Inter Standard AI Encoding Hub

- Written in OCaml ($\simeq$ 9100 LOC)
- Input: ONNX neural networks (universal representation)
- Output: SMTLIB2 targetting several theories (QF_NRA, QF_LRA, QF_FP)
- Under LGPLv2 license
- Heavy use of ppx features
- Abstract API for easy addition of new solvers
- Limitation: no support for generic simulator description

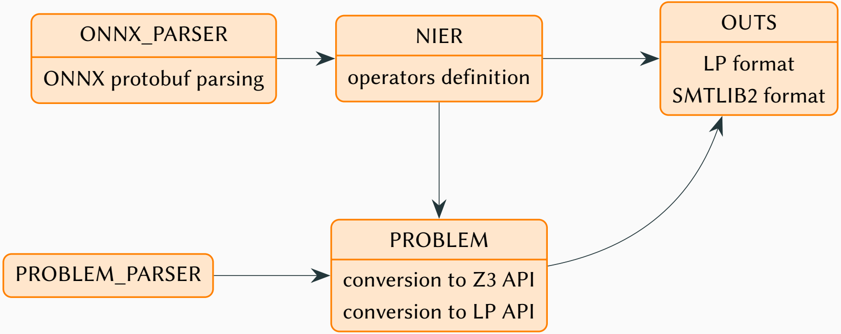
Principle of ISAIEH

Build SMT formulae encoding:

1. Network control flow ϕ^n: flattened and written as SMTLIB2 commands
2. Property to verify ϕ^p
3. Input constraints ϕ^x: linear constraints
4. Simulator description ϕ^g

ISAIEH then sends $\phi^n \wedge \phi^p \wedge \phi^x \wedge \phi^g$ to external solvers

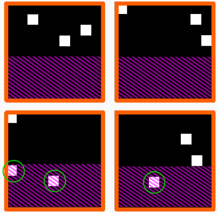
ISAIEH


ONNX_PARSER
ONNX protobuf parsing
NIER
operators definition
OUTS
LP format
SMTLIB2 format
PROBLEM_PARSER
PROBLEM
conversion to Z3 API
conversion to LP API

Synthetic experiment: a simple self driving car perceptive unit

Train a simple model to output a single command directive if a simplified input is in a pre-defined danger zone

$s =$ (position of obstacles)



x

output scalar (obstacle detected if > 0)

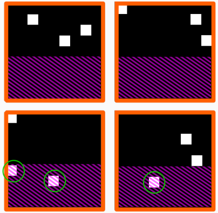
y

Network is relatively small

Synthetic experiment: a simple self driving car perceptive unit

Train a simple model to output a single command directive if a simplified input is in a pre-defined danger zone

$s =$ (position of obstacles)



x

output scalar (obstacle detected if > 0)

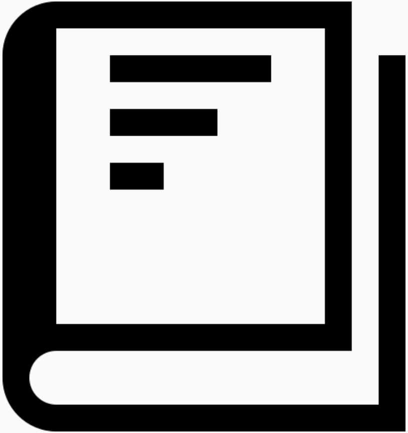
y

Network is relatively small

We have proven the given trained network will ***never fail***

The tooling problem

Specification ✓

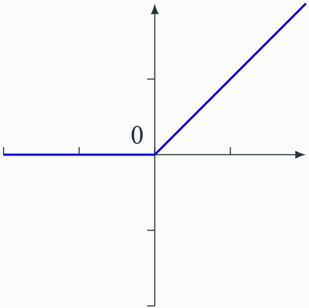


We could rely on simulators to obtain **specifications** for deep learning software

Tooling

How to improve the machinery of traditional solvers to **scale** on deep learning software?

The relu: a piece-wise linear activation function

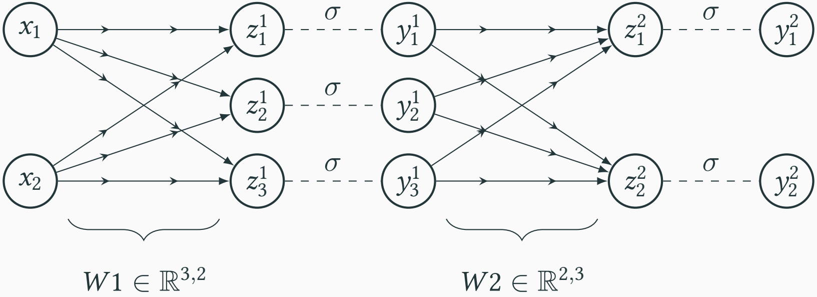


$$relu : x \in \mathbb{R} \mapsto \max(x, 0)$$

relu function, linear on $]-\infty; 0]$ and $[0; \infty[$

$\sigma : x \mapsto relu(x)$ yields two states: either ***active*** $(x > 0)$ or ***inactive*** $(x \leq 0)$

Some notions of deep neural networks



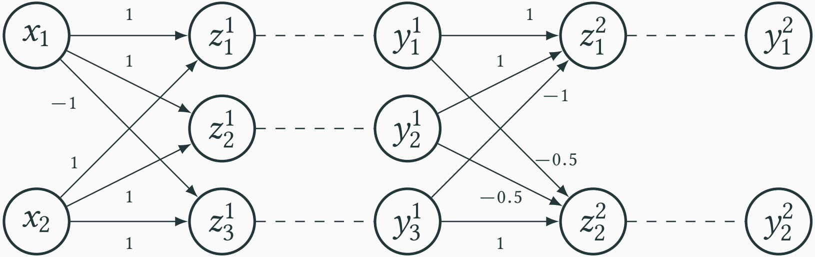
A neural network is a succession of linear operations (addition, multiplication by a constant) followed by a non-linear *activation* function σ

Networks with relu are widely used: we will study them in the rest of this thesis

Feedforward propagation by example

$x_1 \in [0.6, 1]$

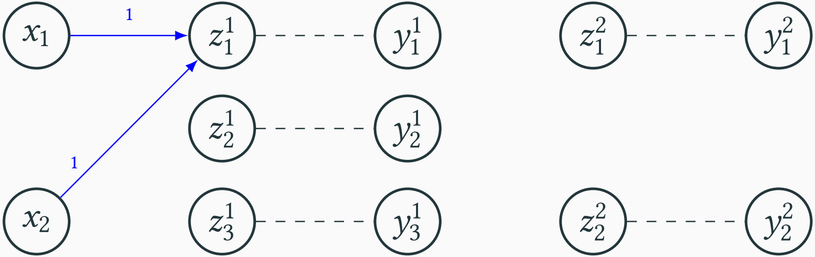
numbers are weights



$x_2 \in [0, 0.4]$

Feedforward propagation by example

$x_1 \in [0.6, 1]$

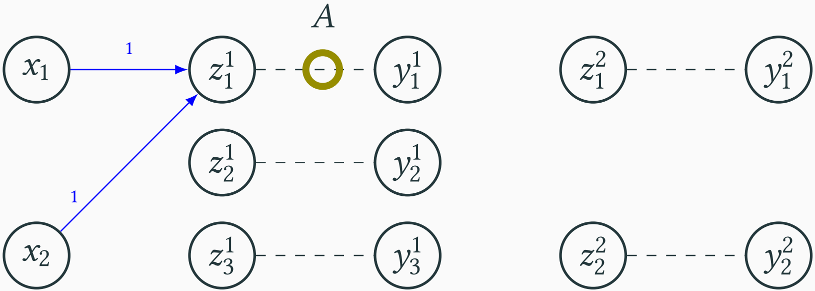


$x_2 \in [0, 0.4]$

$z_1^1 = x_1 + x_2$

Feedforward propagation by example

$x_1 \in [0.6, 1]$

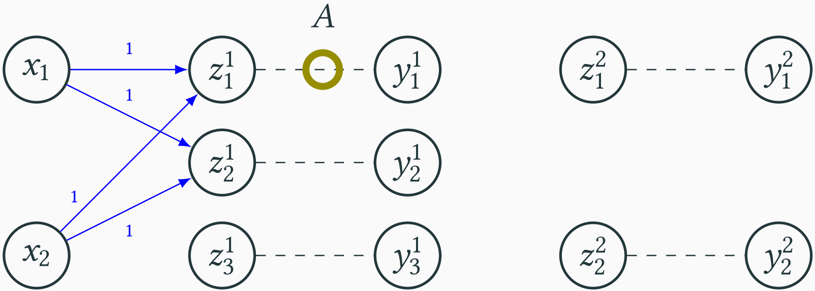


$x_2 \in [0, 0.4]$

$z^1_1 = x_1 + x_2 > 0$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



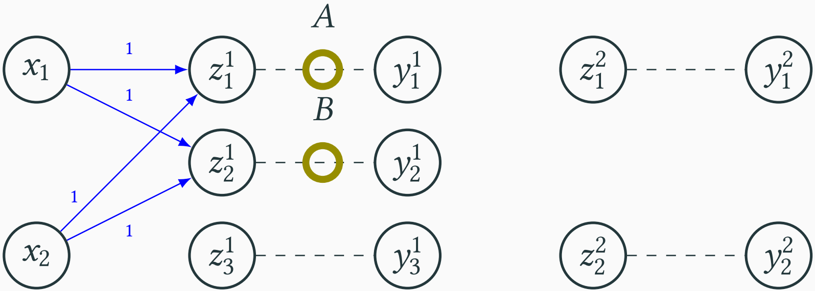
$x_2 \in [0, 0.4]$

$z^1_1 = x_1 + x_2 > 0$

$z^1_2 = x_1 + x_2$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



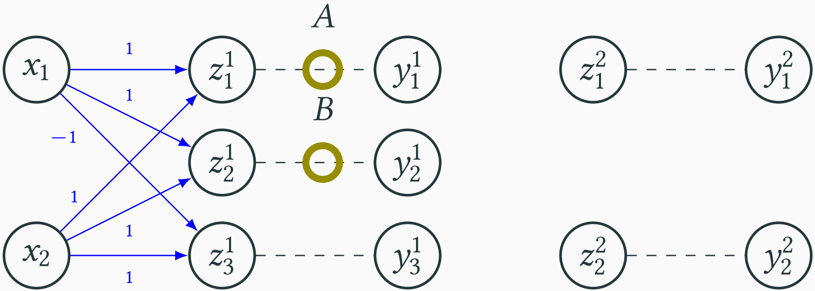
$x_2 \in [0, 0.4]$

$$z_1^1 = x_1 + x_2 > 0$$

$$z_2^1 = x_1 + x_2 > 0$$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

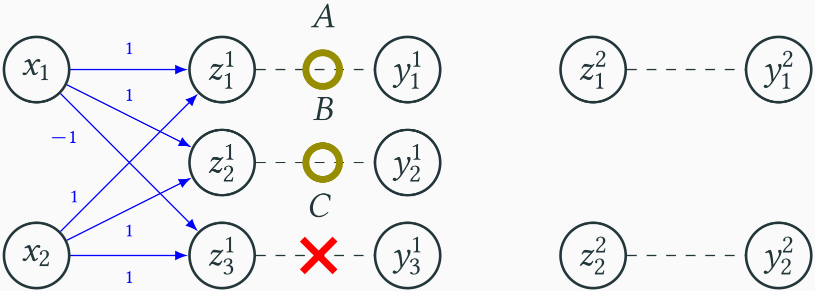
$$z_1^1 = x_1 + x_2 > 0$$

$$z_2^1 = x_1 + x_2 > 0$$

$$z_3^1 = -x_1 + x_2$$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

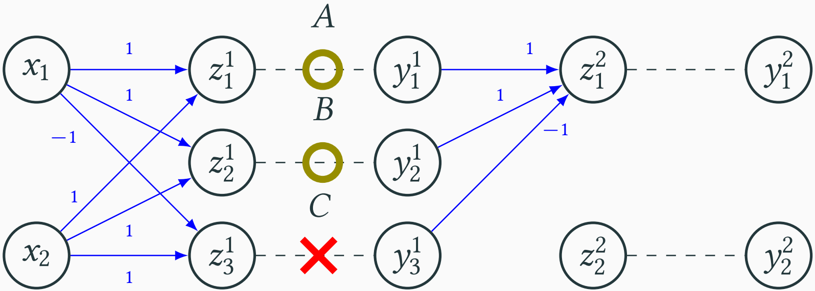
$z_1^1 = x_1 + x_2 > 0$

$z_2^1 = x_1 + x_2 > 0$

$z_3^1 = -x_1 + x_2 < 0$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$z_1^1 = x_1 + x_2 > 0$

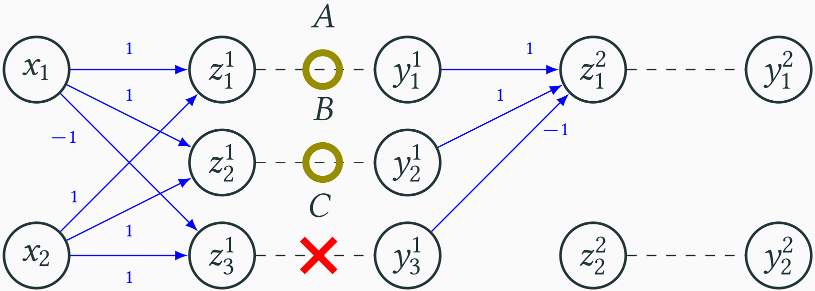
$z_2^1 = x_1 + x_2 > 0$

$z_3^1 = -x_1 + x_2 < 0$

$z_1^2 = y_1^1 + y_2^1 - y_3^1$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$z_1^1 = x_1 + x_2 > 0$

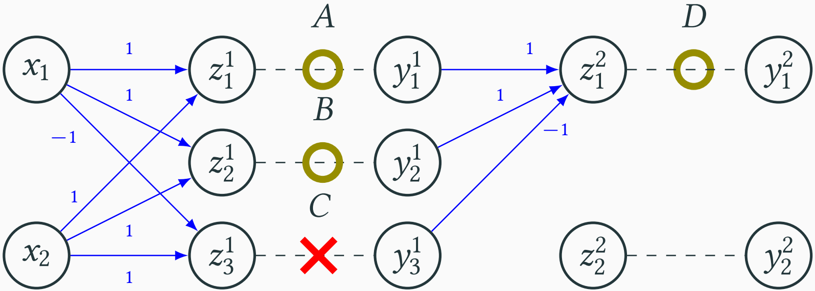
$z_2^1 = x_1 + x_2 > 0$

$z_3^1 = -x_1 + x_2 < 0$

$z_1^2 = 2x_1 + 2x_2$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$z_1^1 = x_1 + x_2 > 0$

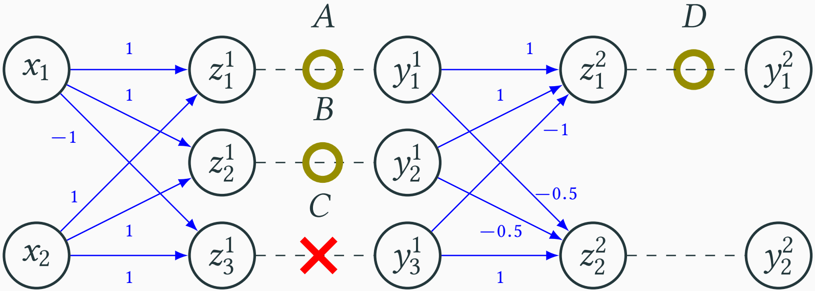
$z_2^1 = x_1 + x_2 > 0$

$z_3^1 = -x_1 + x_2 < 0$

$z_1^2 = 2x_1 + 2x_2 > 0$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$$z_1^1 = x_1 + x_2 > 0$$

$$z_2^1 = x_1 + x_2 > 0$$

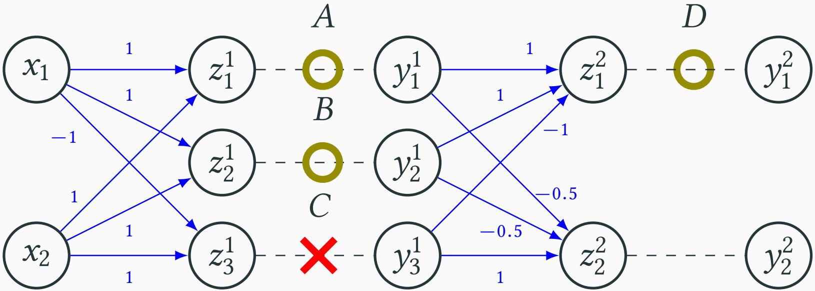
$$z_3^1 = -x_1 + x_2 < 0$$

$$z_1^2 = 2x_1 + 2x_2 > 0$$

$$z_2^2 = -0.5y_1^1 - 0.5y_2^1 + y_3^1$$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$z^1_1 = x_1 + x_2 > 0$

$z^1_2 = x_1 + x_2 > 0$

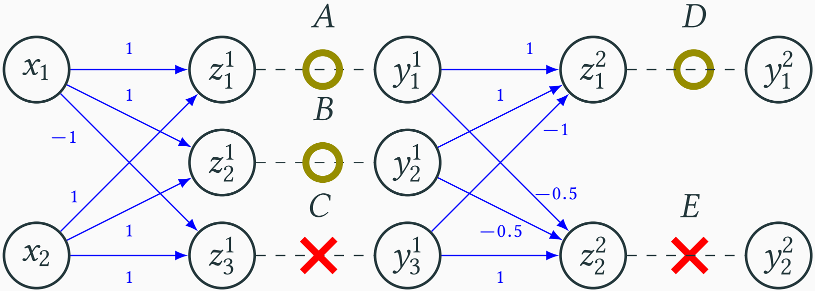
$z^1_3 = -x_1 + x_2 < 0$

$z^2_1 = 2x_1 + 2x_2 > 0$

$z^2_2 = -x_1 - x_2$

Feedforward propagation by example

$x_1 \in [0.6, 1]$



$x_2 \in [0, 0.4]$

$z_1^1 = x_1 + x_2 > 0$

$z_2^1 = x_1 + x_2 > 0$

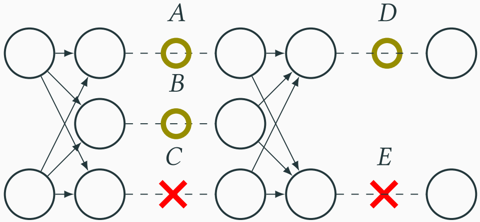
$z_3^1 = -x_1 + x_2 < 0$

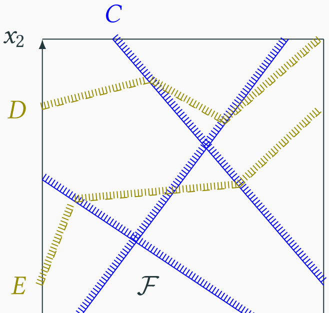
$z_1^2 = 2x_1 + 2x_2 > 0$

$z_2^2 = -x_1 - x_2 < 0$

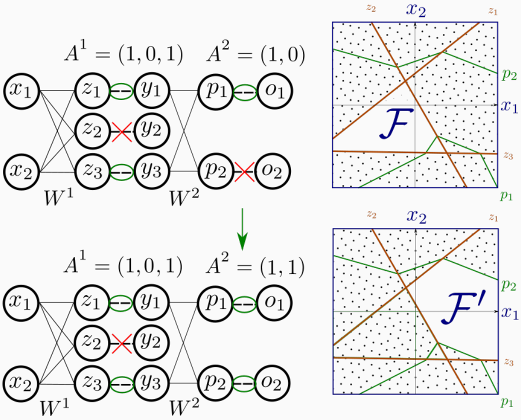
Activation states yield constraints on the input space

1. Activation states result on constraints that partition the input space
2. Activation states of layer l constraint activation states of layers $l+1$, hence the broken lines
3. We call activation regions $\mathcal{F}$ *facets*





Restricting neural networks to facets results in a linear function



The restriction of a network on $\mathcal{F}$ can be rewritten as a linear function:

$$f_{|\mathcal{F}} = diag(A^2)W^2diag(A^1)W^1$$

Current state of affair for specialized tools

1. Formal verification of feedforward relu networks is a NP-complete problem[2]
2. Naive branching at each activation node on a network with n neurons would lead to 2^n cases: combinatorial explosion
3. Prior experiments done with Frama-C EVA showed scalability difficulties on small networks

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

Some hope for the future

1. SAT is a NP-complete problem, but multiple inventions led to a number of highly performant tools (CDCL, 2-watched literals…)
2. Specialized branch-and-bound approaches are starting to get leverage[3]
3. Tighter upper bounds in the number of facets for certain class of networks[4]: $\mathcal{O}(\frac{n^d}{d!})$
4. Neural networks we consider are highly connected, without loops: better search heuristics may arise

[3] Branch and bound for piecewise linear neural network verification, Bunel et al., JMLR 2020

[4] Deep ReLU Networks Have Surprisingly Few Activation Patterns, Hanin et al., NeurIPS 2019

Neural networks are linear functions when restricted to a facet

Neural networks are linear functions when restricted to a facet

Linear functions are more amenable for solvers

Neural networks are linear functions when restricted to a facet

Linear functions are more amenable for solvers

Enumerating facets and verifying properties on each may be scalable

Neural networks are linear functions when restricted to a facet

Linear functions are more amenable for solvers

Enumerating facets and verifying properties on each may be scalable

DISCO Verification: Division of Input Space into COnvex polytopes for neural network verification[5]

[5] *Partitionnement en régions linéaires pour la vérification formelle de réseaux de neurones*, Girard-Satabin, Varasse et al., JFLA 2021

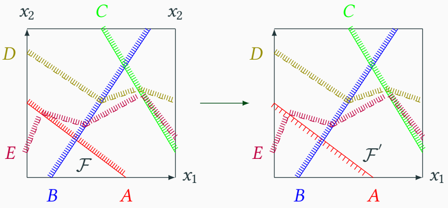
How to enumerate facets?

First attempt was a geometric approach: from one facet, find the geometrical neighbours. Vertex Enumeration is a well-researched problem

[6] The quickhull algorithm for convex hulls, Barber et al., ACM Transactions on Mathematical Software, 4 Dec. 1996

How to enumerate facets?

First attempt was a geometric approach: from one facet, find the geometrical neighbours. Vertex Enumeration is a well-researched problem

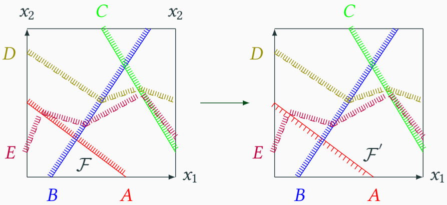


- high dimensional geometry (curse of dimensionality)
- dependency between layers $\implies$ no vertex enumeration
- complexity for convex hull for one facet[6] is $\mathcal{O}(\frac{n^{d/2}}{d/2!})$

[6] The quickhull algorithm for convex hulls, Barber et al., ACM Transactions on Mathematical Software, 4 Dec. 1996

How to enumerate facets?

First attempt was a geometric approach: from one facet, find the geometrical neighbours. Vertex Enumeration is a well-researched problem



- high dimensional geometry (curse of dimensionality)
- dependency between layers $\implies$ no vertex enumeration
- complexity for convex hull for one facet[6] is $\mathcal{O}(\frac{n^{d/2}}{d/2!})$

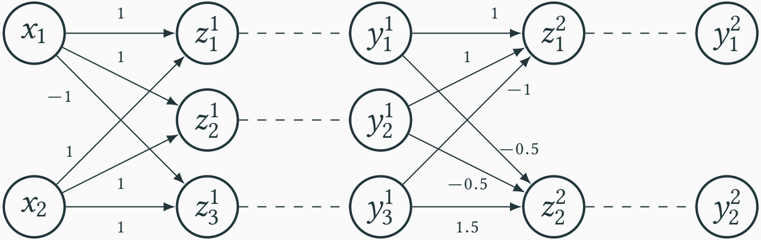
Implementation should follow another path

[6] The quickhull algorithm for convex hulls, Barber et al., ACM Transactions on Mathematical Software, 4 Dec. 1996

DISCO by example

$x_1 \in [0, 1]$

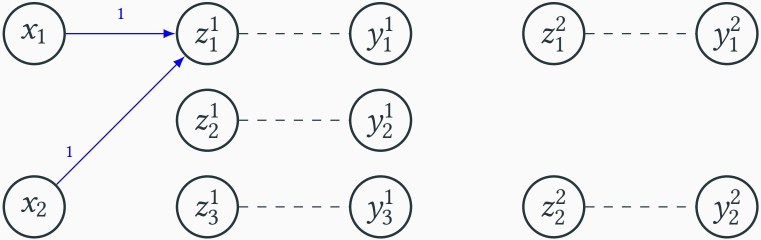
$x_2 \in [0, 1]$



DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$

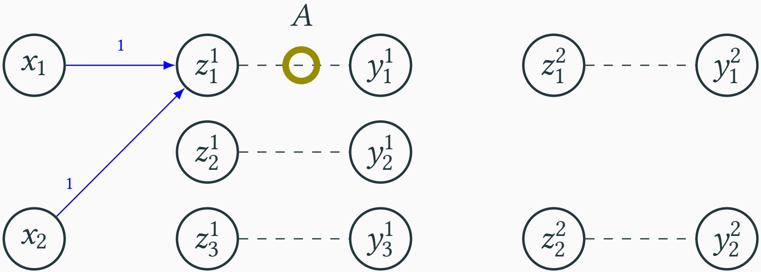


$$z_1^1 = z_2^1 = x_1 + x_2 > 0$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



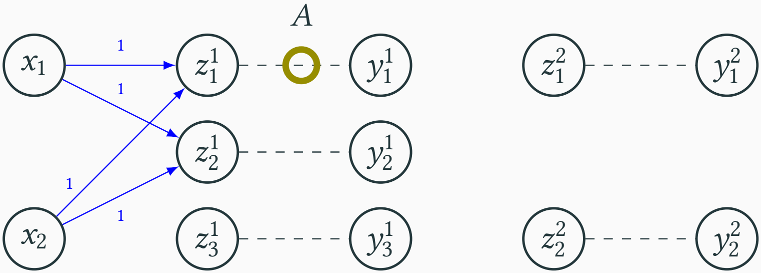
$$z_1^1 = z_2^1 = x_1 + x_2 > 0$$

$$y_1^1 = z_1^1$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



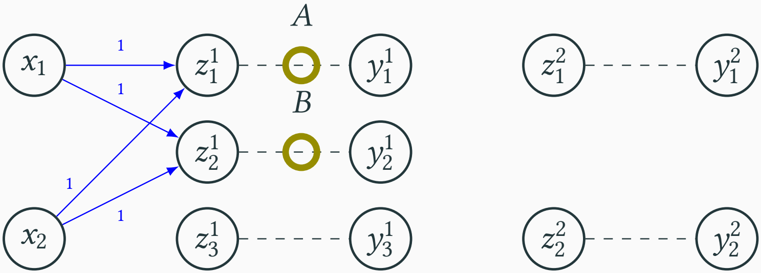
$$z_1^1 = z_2^1 = x_1 + x_2 > 0$$

$$y_1^1 = z_1^1$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$

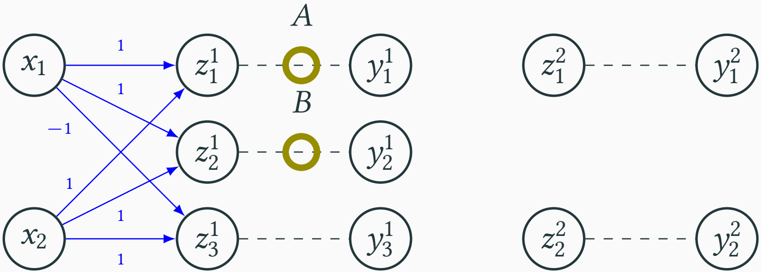


$$\begin{aligned} z^1_1 = z^1_2 = x_1 + x_2 > 0 \\ y^1_1 = z^1_1 \\ y^1_2 = z^1_2 \end{aligned}$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$

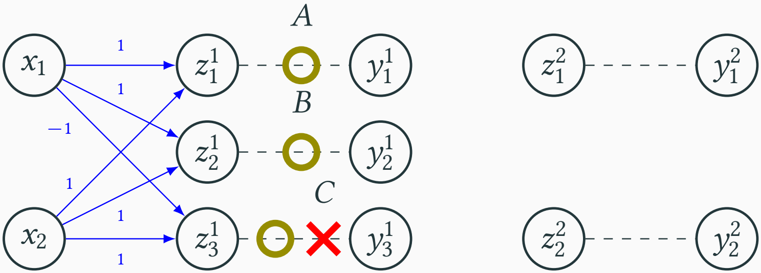


$$z_1^1 = z_2^1 = x_1 + x_2 > 0$$
$$y_1^1 = z_1^1$$
$$y_2^1 = z_2^1$$
$$z_3^1 = -x_1 + x_2$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$
\begin{aligned}
z_1^1 = z_2^1 = x_1 + x_2 > 0 \\
y_1^1 = z_1^1 \\
y_2^1 = z_2^1 \\
z_3^1 = -x_1 + x_2
\end{aligned}
$$

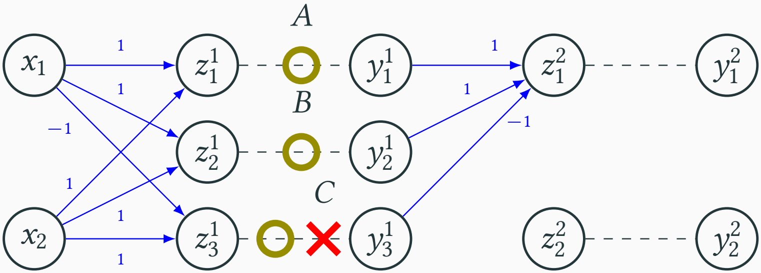
$z_3^1 \geq 0 \qquad y_3^1 = z_3^1$

$z_3^1 < 0 \qquad y_3^1 = 0$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$
\begin{aligned}
z_1^1 = z_2^1 &= x_1 + x_2 > 0 \\
y_1^1 &= z_1^1 \\
y_2^1 &= z_2^1 \\
z_3^1 &= -x_1 + x_2 \\
z_1^2 &= y_1^1 + y_2^1 - y_3^1
\end{aligned}
$$

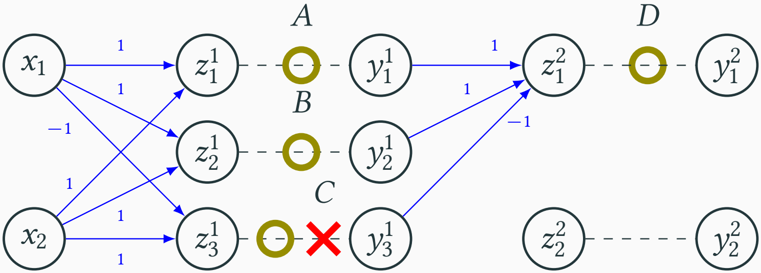
$z_3^1 \geq 0$ $y_3^1 = z_3^1$

$z_3^1 < 0$ $y_3^1 = 0$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$\begin{aligned} z_1^1 = z_2^1 &= x_1 + x_2 > 0 \\ y_1^1 &= z_1^1 \\ y_2^1 &= z_2^1 \\ z_3^1 &= -x_1 + x_2 \\ z_1^2 &= y_1^1 + y_2^1 - y_3^1 \end{aligned}$$

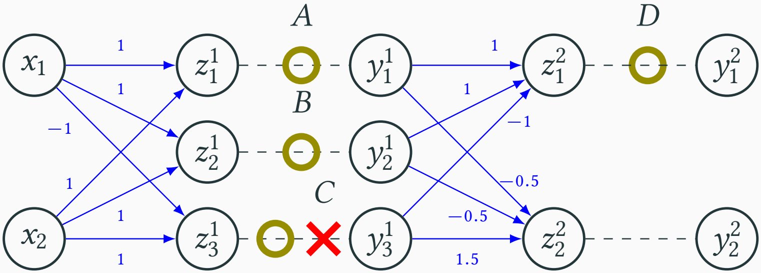
$$\begin{aligned} z_3^1 &\geq 0 & y_3^1 &= z_3^1 \\ z_1^2 &\geq 0 & y_1^2 &= z_1^2 \end{aligned}$$

$$\begin{aligned} z_3^1 &< 0 & y_3^1 &= 0 \\ z_1^2 &\geq 0 & y_1^2 &= z_1^2 \end{aligned}$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$
\begin{aligned}
z_1^1 = z_2^1 &= x_1 + x_2 > 0 \\
y_1^1 &= z_1^1 \\
y_2^1 &= z_2^1 \\
z_3^1 &= -x_1 + x_2 \\
z_1^2 &= y_1^1 + y_2^1 - y_3^1 \\
z_2^2 &= -0.5(y_1^1 + y_2^1) + 1.5y_3^1
\end{aligned}
$$

$z_3^1 \geq 0 \quad y_3^1 = z_3^1$

$z_1^2 \geq 0 \quad y_1^2 = z_1^2$

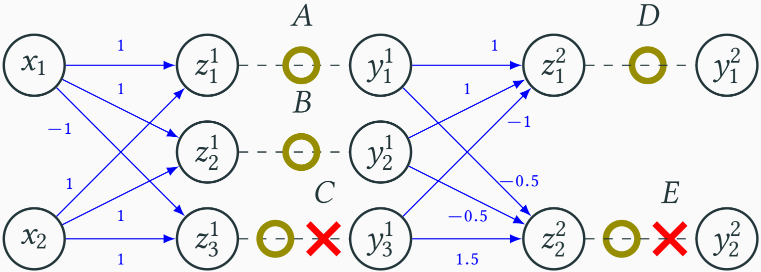
$z_3^1 < 0 \quad y_3^1 = 0$

$z_1^2 \geq 0 \quad y_1^2 = z_1^2$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$\begin{aligned}
z_1^1 = z_2^1 &= x_1 + x_2 > 0 \\
y_1^1 &= z_1^1 \\
y_2^1 &= z_2^1 \\
z_3^1 &= -x_1 + x_2 \\
z_1^2 &= y_1^1 + y_2^1 - y_3^1 \\
z_2^2 &= -0.5(y_1^1 + y_2^1) + 1.5y_3^1
\end{aligned}$$

$$\begin{aligned}
z_3^1 &\geq 0 & y_3^1 &= z_3^1 \\
z_1^2 &\geq 0 & y_1^2 &= z_1^2 \\
z_2^2 &< 0 & y_2^2 &= 0
\end{aligned}$$

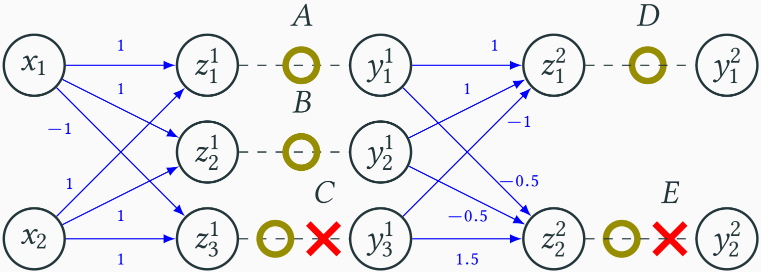
$$\begin{aligned}
z_3^1 &< 0 & y_3^1 &= 0 \\
z_1^2 &\geq 0 & y_1^2 &= z_1^2 \\
z_2^2 &< 0 & y_2^2 &= 0
\end{aligned}$$

$$\begin{aligned}
z_3^1 &\geq 0 & y_3^1 &= z_3^1 \\
z_1^2 &\geq 0 & y_1^2 &= z_1^2
\end{aligned}$$

DISCO by example

$x_1 \in [0, 1]$

$x_2 \in [0, 1]$



$$\begin{aligned} z_1^1 = z_2^1 &= x_1 + x_2 > 0 \\ y_1^1 &= z_1^1 \\ y_2^1 &= z_2^1 \\ z_3^1 &= -x_1 + x_2 \\ z_1^2 &= y_1^1 + y_2^1 - y_3^1 \\ z_2^2 &= -0.5(y_1^1 + y_2^1) + 1.5 y_3^1 \end{aligned}$$

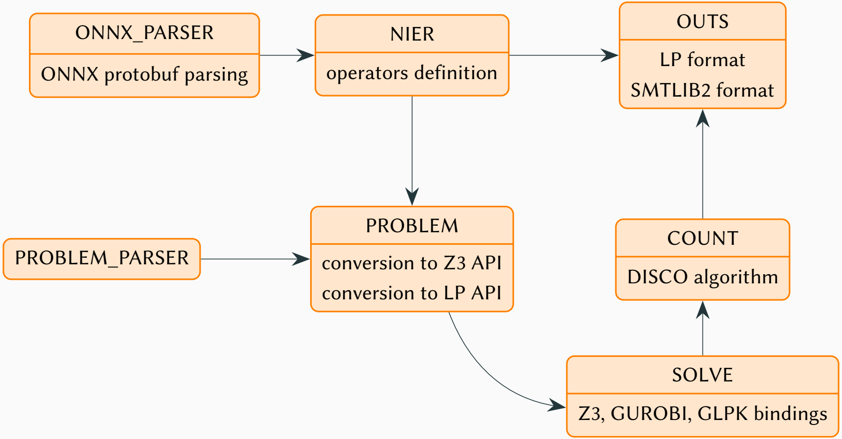
$$\begin{aligned} z_3^1 &\geq 0 & y_3^1 &= z_3^1 \\ z_1^2 &\geq 0 & y_1^2 &= z_1^2 \\ z_2^2 &< 0 & y_2^2 &= 0 \end{aligned}$$

$$\begin{aligned} z_3^1 &< 0 & y_3^1 &= 0 \\ z_1^2 &\geq 0 & y_1^2 &= z_1^2 \\ z_2^2 &< 0 & y_2^2 &= 0 \end{aligned}$$

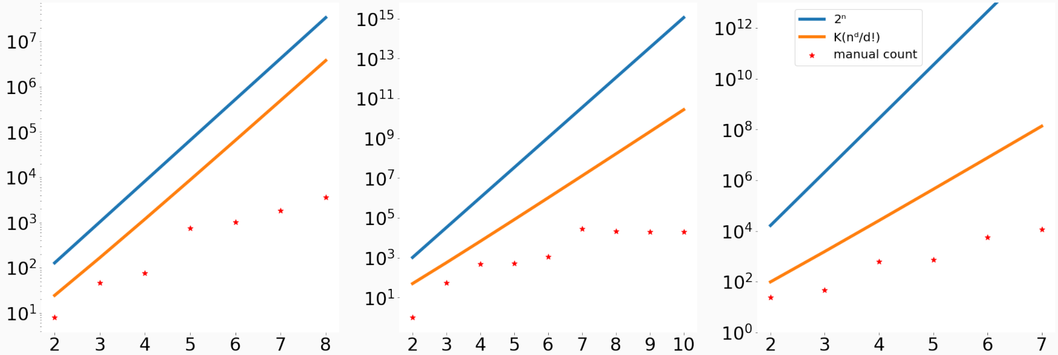
$$\begin{aligned} z_3^1 &\geq 0 & y_3^1 &= z_3^1 \\ z_1^2 &\geq 0 & y_1^2 &= z_1^2 \end{aligned}$$

Stacks describe facets and linear operations

ISAIEH unveiled

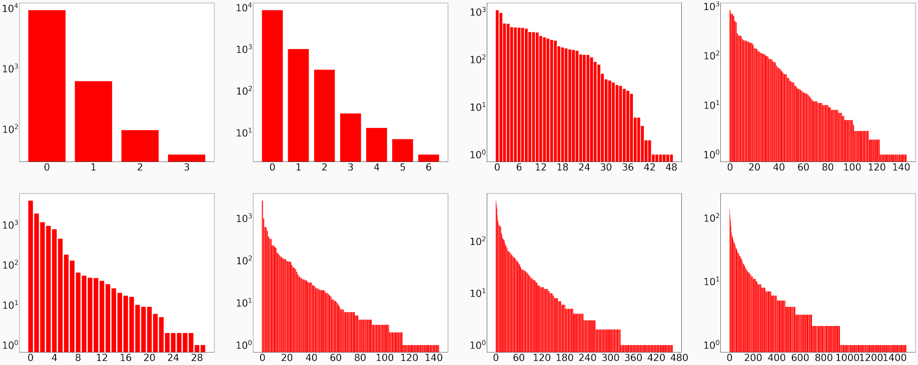


Comparing to upper bounds



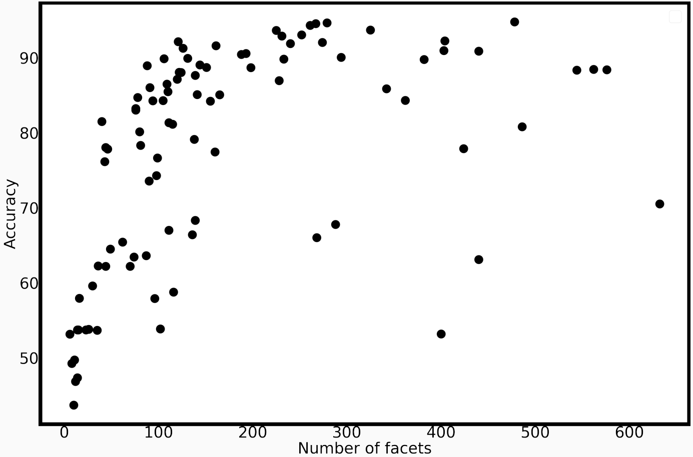
Number of facets for several input dimensions

Facet predominance

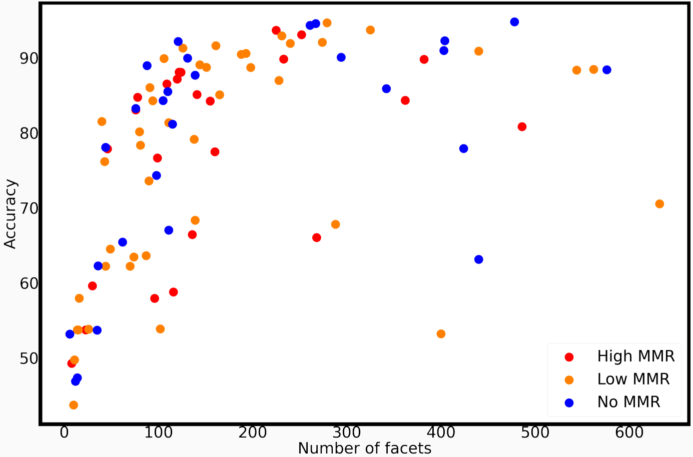


Number of points per unique activation pattern for 10000 samples

A tradeoff between accuracy and number of facets



A tradeoff between accuracy and number of facets



~~Maximum Margin Regularization[7] (MMR) does not have a~~ significant impact

[7] Provable Robustness of ReLU networks via Maximization of Linear Regions, Croce et al., AISTATS 2019

Runtimes

Problem	No split	DISCO verification	Facet enumeration	Total time DISCO
multiplication 3	0.769s±0.0205	**0.145s**±0.012	2.69s±0.0596	2.83s
multiplication 4	5.43s±0.31	**0.71s**±0.0591	13.1s±0.859	13.8s
multiplication 5	**0.0179s**±0.00596	0.0771s±0.0077	0.699s±0.0124	0.776s
multiplication 6	**0.0264s**±0.00124	0.988s±0.0693	11.6s±0.186	12.6s
multiplication 7	**0.0474s**±0.00158	16.8s±0.831	227s±8.51	244s
multiplication 8	**0.0484s**±0.00551	1.65s±0.113	27.2s±0.576	28.8s
5 × 5 perception	132s	23.7s	0.86s	**24.56s**
7 × 7 perception	TIMEOUT (>10000s)	1393s	15.38s	**1406.38s**

DISCO boost SMT solvers **without changing their inner working** (can be further enhanced with heuristics)

Related work

1. Marabou[8] relies on SMT solving to perform branch and bound, but is still specialized against ACAS benchmarks
2. ERAN[9] is much faster than DISCO (less than 0.01s on perception), but can only handle linear properties
3. Facets enumeration algorithm exists[10], but not for formal verification

DISCO is **slower**, but **more generic** and **solver agnostic**

[8] Katz et al., 2019

[9] Vechev et al., 2018-2021

[10] Serra et al., 2018

Conclusion

Scientific contributions

- Proposed a formalism to use simulators as perceptual inputs specification
- Implemented an ONNX to SMTLIB and LP compiler
- Built a problem splitting algorithm taking advantage of the piecewise linear nature of relu networks
- Analyzed linear regions distribution and explored their practical use for formal verification of deep neural networks

Perspectives

1. Simulators for ISAIEH: taking into account simulators as machine learning programs
2. Enhancing DISCO: counting facets directly is not the best approach; using dependency analysis to reduce the number of solver calls
3. Engineering work to deal with more design possibilites (cambrian explosion of tools)
4. Expressing more complex properties is key: industrial adoption is the goal

Contributions

Publications and prepublications

- *DISCO Verification: Division of Input Space into COnvex polytopes for neural network verification*, J. Girard-Satabin, A. Varasse, G. Charpiat, Z. Chihani, M. Schoenauer, to be published
- *Partitionnement en régions linéaires pour la vérification formelle de réseaux de neurones*, J. Girard-Satabin, A. Varasse, G. Charpiat, Z. Chihani, M. Schoenauer, JFLA 2021
- *CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators*, J. Girard-Satabin, G. Charpiat, Z. Chihani, M. Schoenauer, ECAI 2020

Dissemination

- *Theory and practice of deep neural network verification*, DFKI 2021, PFIA 2020, also as a M2 course at Master SETI
- *Detection of behaviours using machine learning in the public space*, La Quadrature du Net, outreach conference, 2021
- *ForMaL DIGICOSME Spring School*, 2019

Software

- Inter Standard AI Ecoding Hub (ISAIEH), LGPLv2, to be merged within the CAISAR platform developped at LSL