Building specifications for perception systems : formal proofs of deep networks trained with simulators

Julien Girard-Satabin (CEA LIST), Guillaume Charpiat (INRIA TAU), Zakaria Chihani (CEA LIST), Marc Schoenauer (INRIA TAU)

10 septembre 2019

Outline

Necessity to certify deep neural networks and challenges

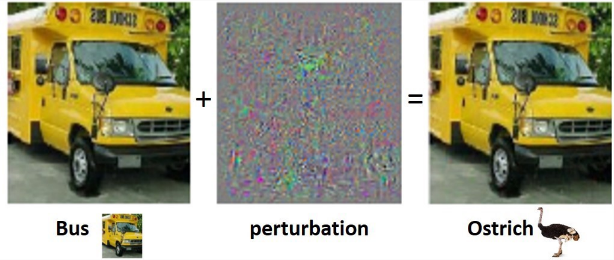
Deep neural networks are awesome. . .

Active research community, profusion of tools, lot of industrial applications. . .

Deep neural networks are awesome. . .

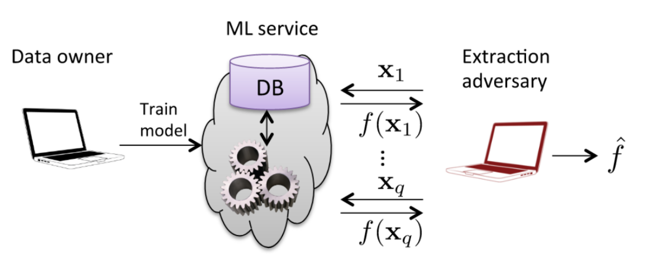
Active research community, profusion of tools, lot of industrial applications. . . . . . yet they are not perfect

Adversarial examples (Szegedy et al. 2013)

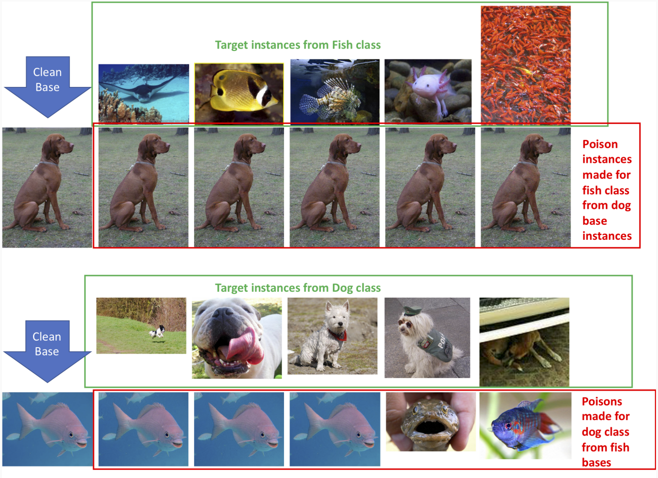


Innocuous to humans, transferable between datasets, not systematic detection method

Model theft (Tramèr et al. 2018)



Dataset poisoning (Shafahi et al. 2018)

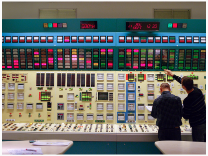


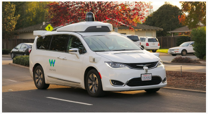
Context : critical systems

A critical system is a system which failure may cause physical harm, economical losses or damage the environment

Context : critical systems

A critical system is a system which failure may cause physical harm, economical losses or damage the environment





How to bring confidence in software systems ?

Goal : guarantee that the system respects a *safety specification*

ϕ : *an autonomous car will not roll over pedestrians*

What about tests ?

Test on real environment	real conditions	cumbersome, potentially hazardous, non exhaustive
Test on virtual environment	can be automated, easy to integrate in existing workflow	non exhaustive, biased towards success

And more (fuzzing. . .)

Tests alone are not enough !

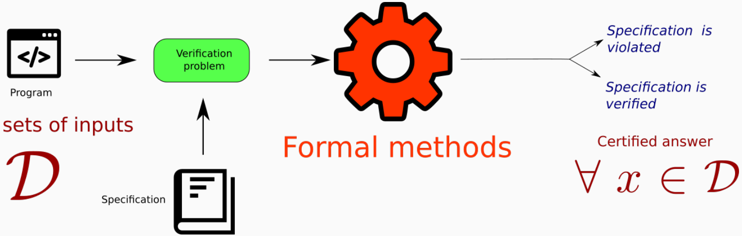
Claim	Discussion
"A car drove 5,472km, 99% in autonomous mode"[1]	If it translate to a failure rate, 10^{-2} , insufficient compared to requirements in other critical systems (about 10^{-8} in aerospace)
"Our test cases are exhaustive"	Testing sets tend to be biased towards "normal" operation (accidents are rare)[2] 

1. https ://www.wired.com/2015/04/delphi-autonomous-car-cross-country/
2. https ://arstechnica.com/cars/2019/05/feds-autopilot-was-active-during-deadly-march-tesla-crash/

Introducing formal methods

- Studied in the academics since 1930 ($\lambda-$calculus, Church, Turing)
- Different techniques : abstract interpretation (Cousot and Cousot 1977), SAT/SMT (Davis and Putman 1960 ; Tinelli 2009), deductive verification (Coquand 1989), etc.
- Used in industrial settings such as aerospace, automated transports, energy to *formally* certify

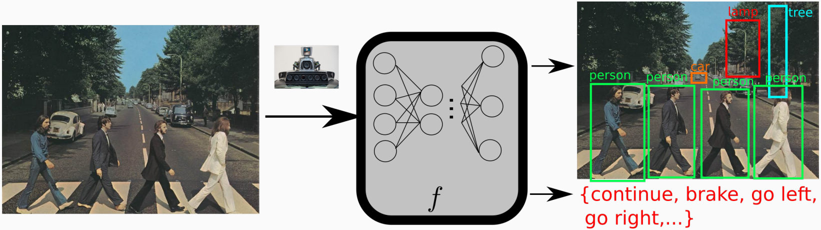
Key points



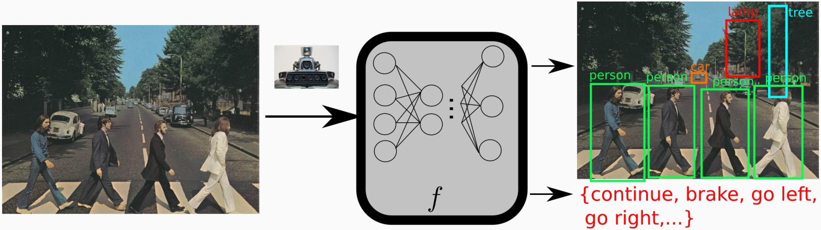
Work on domains $\mathcal{D}$ of inputs (global properties)

Answer is sound, formally guaranteed

Case study : a self-driving car perception unit

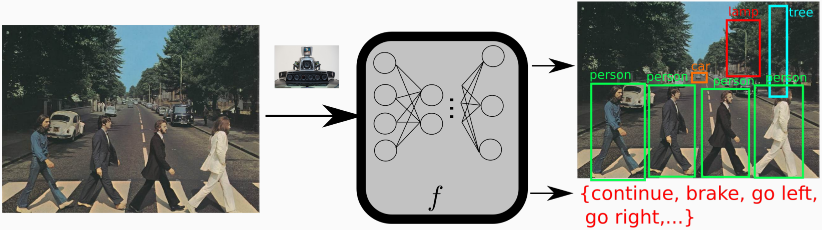


Case study : a self-driving car perception unit



Dream property ϕ : *the autonomous car never roll over pedestrians*

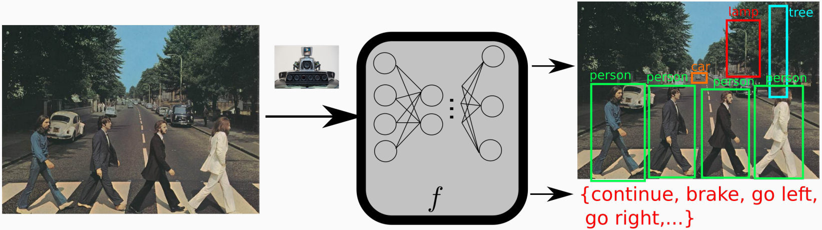
Case study : a self-driving car perception unit



Dream property ϕ : *the autonomous car never roll over pedestrians*

no *formal characterization* of what a pedestrian is !

Case study : a self-driving car perception unit



Dream property ϕ : *the autonomous car never roll over pedestrians*

no *formal characterization* of what a pedestrian is !

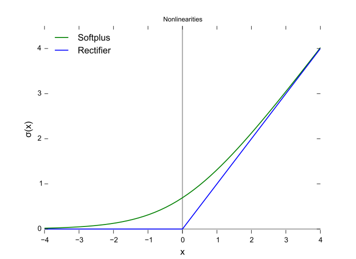
Lack of formal definition on inputs prevent from formulating interesting safety properties

It's hard to use formal methods on deep learning

Classical software	Machine learning
Explicit control flow	Generated control flow
Explicit specifications	Data-driven specifications (lack of generality)
Abstractions and well known concepts	Very few abstractions and reusability
Documented and understood vulnerabilites	Flaws without systematic characterization

Some differences between classical software and machine learning

Another difficulty : performance of verification tools



2 cases per ReLU node for the solvers

Several million ReLU nodes

$\rightarrow 2^{O(10^6)}$ case splits

Combinatory explosion (if done naively)

Deep learning verification : a review

Local properties : adversarial robustness

For a *given* input $\mathbf{x}$, a classification function f, an adversarial perturbation δ :

find delta satisfying

classifier misclassification

such that

perturbation stays below a certain threshold

Local properties : adversarial robustness

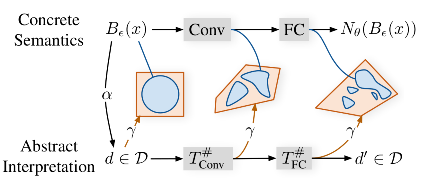
For a *given* input **x**, a classification function f, an adversarial perturbation δ :

find delta satisfying $f(x) \neq f(x+\delta)$

such that $\|\delta\|_p \leq \varepsilon$

DiffAI/DeepPoly (Gehr et al. 2018, Singh et al. 2019)

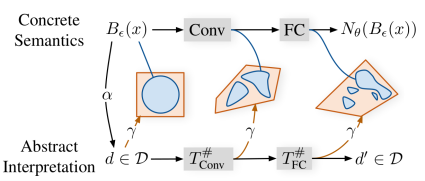
1. *abstract* the network
2. *propagate* perturbations
3. *assess* robustness properties
4. *learn to minimize* adversarial loss



Improve adversarial robustness on 100 samples from CIFAR-10 from 0 to 80%, $\varepsilon = 8/255$, 3 hidden layers, convolutional network

DiffAI/DeepPoly (Gehr et al. 2018, Singh et al. 2019)

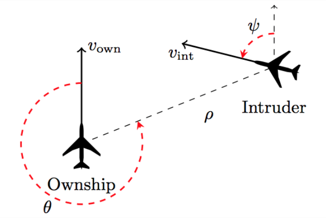
1. *abstract* the network
2. *propagate* perturbations
3. *assess* robustness properties
4. *learn to minimize* adversarial loss



Improve adversarial robustness on 100 samples from CIFAR-10 from 0 to 80%, $\varepsilon = 8/255$, 3 hidden layers, convolutional network

Scalable verification, but local properties

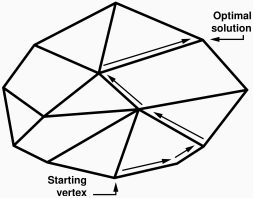
Global properties : ACAS-Xu



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. Bounds : $\rho \geq 55947.691, v_{own} \geq 1145, v_{int} \leq 60$

Critical system

ReLuPlex and Marabou (Katz et al. 2017, Katz et al. 2019)



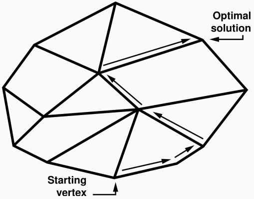
Core of most SMT solvers working with number values

Modified to lazily evaluate ReLUs

Exact verification of several properties on a ACAS-Xu implementation

Global properties

ReLuPlex and Marabou (Katz et al. 2017, Katz et al. 2019)



Core of most SMT solvers working with number values

Modified to lazily evaluate ReLUs

Exact verification of several properties on a ACAS-Xu implementation

Global properties

Prior *model* for the inputs, assuming the detection is perfect. How do we verify perception ? What is an intruder ?

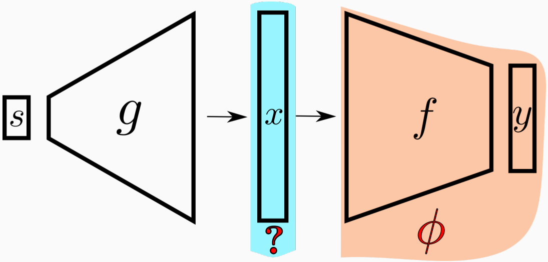
Verification of perception models trained with simulators

Example of simulator

Industry rely more and more on simulators to generate scenarios to train and evaluate deep learning models

Screenshot from the CARLA open source simulator

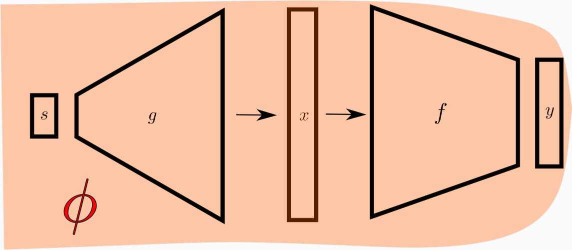
Simulators as data providers



- s : parameters (obstacles, weather conditions...)
- g : simulator
- ϕ : "$\forall$ x that contains a pedestrian, do not roll over it"
- x : perceptual input (images)
- f : model
- y : decision output (brake...)

How to formulate ϕ ? What is a pedestrian in x ?

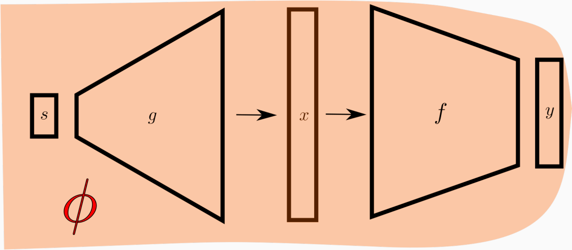
Reformulation of our verification problem



Modify the verification problem formulation to include g and s

ϕ now encompass s and can now be expressed : *For certain values of s that can be translated by g as the presence of pedestrians into x, do not run over pedestrians*

Reformulation of our verification problem

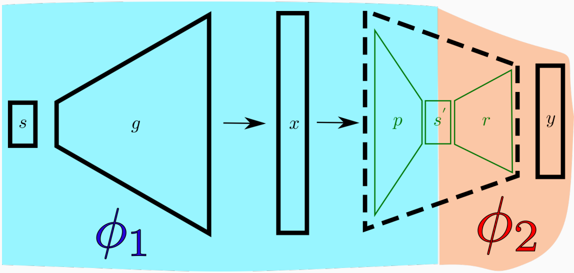


Modify the verification problem formulation to include *g* and *s*

ϕ now encompass *s* and can now be expressed : *For certain values of s that can be translated by g as the presence of pedestrians into x, do not run over pedestrians*

We now have a property to verify a perceptive unit !

Refinement : splitting perception and reasoning

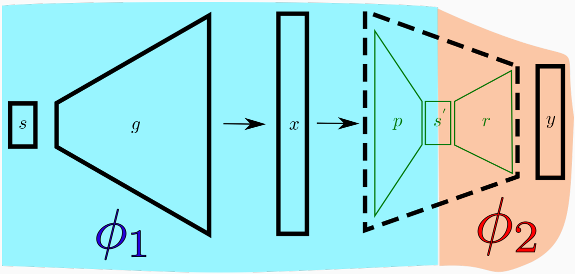


f split in *p*erception and *r*easoning, p learns s

ϕ_1 on p : guarantee of no information loss : *reconstruct s from x*
$s' = s \ \forall \ s \rightarrow p \circ g = Id$

ϕ_2 on r : do not kill pedestrians (assuming perfect perception)

Refinement : splitting perception and reasoning



f split in *p*erception and *r*easoning, p learns s

ϕ_1 on p : guarantee of controlled information loss : *reconstruct s from x*
$s^{'} \simeq s \; \forall \; s \rightarrow p \circ g - id < \varepsilon$

ϕ_2 on r : do not kill pedestrians (assuming perfect perception)

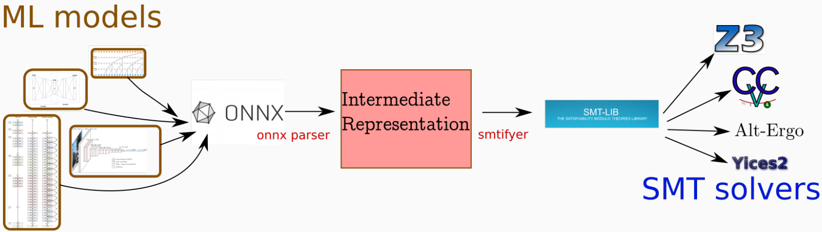
How to achieve that concretely ?

How to express ϕ, g, f, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{S}$?

How to achieve that concretely ?

How to express ϕ, g, f, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{S}$?

SMTLIB : Tinelli et al., 2017, https ://onnx.ai/



High-level workflow

Proof of concept and future works

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

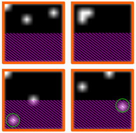
Synthetic experiment : a simple self driving car perceptive unit

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

$(x_1, x_2 \ldots, x_n)$

obstacle position

s



x

output scalar (obstacle detected if > 0)

y

Network has 16 neurons, 2 hidden layers

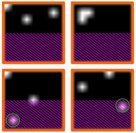
Synthetic experiment : a simple self driving car perceptive unit

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

$(x_1, x_2 \ldots, x_n)$

obstacle position

s



x

output scalar (obstacle detected if > 0)

y

Network has 16 neurons, 2 hidden layers

We prove the given trained network will *never fail*

Translation of model

```
;; Inputs can be only between 0 and 1
(assert (or (= actual_input0000 0) (= actual_input0000 1.)))
(assert (or (= actual_input0001 0) (= actual_input0001 1.)))
(assert (or (= actual_input0002 0) (= actual_input0002 1.)))
(assert (or (= actual_input0003 0) (= actual_input0003 1.)))
. . .
;; There is at least one input within the danger zone (pixels from 35)
;; that is equal to 1
(assert
    (or
        (or (= actual_input0035 1.)
        (= actual_input0036 1.))
        (or (= actual_input0037 1.)
        (= actual_input0038 1.))
        . . .
;; Formulate constraint on outputs:
;; The output always fire higher than a
;; confidence value
;; Negation: the output fire lower than a confidence value
(declare-fun confidence () Real)
(assert (= confidence 0.2))
(assert (< |y_out_0_0| confidence))
```

Property formulation for 9x9 input size

```
        . . .
(declare-fun l_1.weight38 () Real)
(assert (= l_1.weight38 (/ (- 13947381) 1208925819614629174706176)))
(declare-fun l_1.weight37 () Real)
(assert (= l_1.weight37 (/ (- 405697 ) 2251799813685248)))
(declare-fun l_1.weight36 () Real)
        . . .
```

Part of the network's translation

Translation of model

```
;; Inputs can be only between 0 and 1
(assert (or (= actual_input0000 0) (= actual_input0000 1.)))
(assert (or (= actual_input0001 0) (= actual_input0001 1.)))
(assert (or (= actual_input0002 0) (= actual_input0002 1.)))
(assert (or (= actual_input0003 0) (= actual_input0003 1.)))
. . .
;; There is at least one input within the danger zone (pixels from 35)
;; that is equal to 1
(assert
    (or
        (or (= actual_input0035 1.)
        (= actual_input0036 1.))
        (or (= actual_input0037 1.)
        (= actual_input0038 1.))
        . . .
;; Formulate constraint on outputs:
;; The output always fire higher than a
;; confidence value
;; Negation: the output fire lower than a confidence value
(declare-fun confidence () Real)
(assert (= confidence 0.2))
(assert (< |y_out_0_0| confidence))
```

Property formulation for 9x9 input size

```
        . . .
(declare-fun l_1.weight38 () Real)
(assert (= l_1.weight38 (/ (- 13947381) 1208925819614629174706176)))
(declare-fun l_1.weight37 () Real)
(assert (= l_1.weight37 (/ (- 405697 ) 2251799813685248)))
(declare-fun l_1.weight36 () Real)
        . . .
```

Part of the network's translation

Problem solved with mainstream SMT solvers

Future work

1. Include noise and incomplete reconstruction in the framework
2. Add rewriting rules
3. Release and enhance the toolkit
4. Add a systematic representation of the simulator
5. Integration of state-of-the art verification tools

Questions ?

Any questions ?