Formal verification of robustness properties in deep learning programs

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

10 février 2020

Formalizing robustness

Classical robustness definition

IEEE Std 610.12-1990 : *"The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions"*

Some examples :

- Sensor noise in embedded systems
- Unvoluntary faulty inputs by the user (unsanitized inputs)

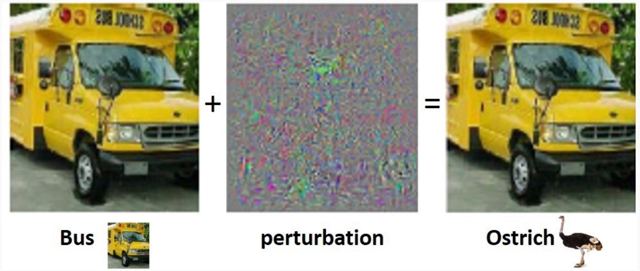
(Oversimplified) classical robustness enforcement process

1. Modeling of environment and faults
2. Various analysis (formal methods, tests) on software to identify sensible failure points
3. Workarounds implementation, redundancy and diversity (multiple functionally similar systems but dissimilar technically), better coding practices, etc.

Neural networks are really specific programs

1. Computer Vision, Natural Language Processing work on highly dimensional, unstructured data
   $\Rightarrow$ **environment modelling difficult** and **scalability issues**
2. Feedforward neural networks are functionally simple (no loops), but variables are meaningless by themselves
   $\Rightarrow$ **current analysis practices not useful**
3. Some very specific failure modes, difficult to spot and even more to fix
   $\Rightarrow$ **faults analysis and correction is impossible for now**

What are adversarial examples ?



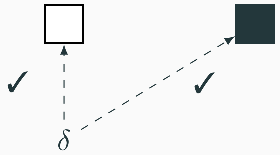
Video for visual adversarial examples (Synthesizing Robust Adversarial Examples, Athalye et al., 2017)

Video for audio adversarial examples (Audio Adversarial Examples: Targeted Attacks on Speech-to-Text, Nicholas Carlini, 2018)

Why are adversarial examples important ?

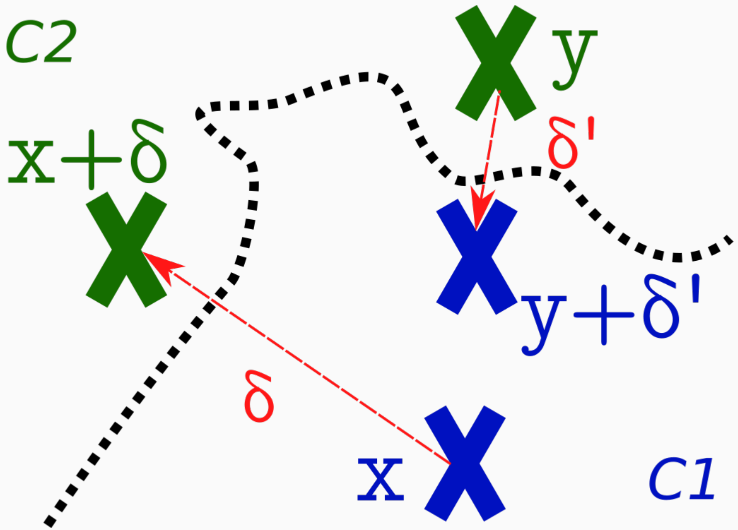
Adversarial examples :

- are transferable (Papernot et al., 2016, Transferability in Machine Learning. . ., Carlini et al. papers)



- not well understood (Adversarial Spheres, Goodfellow et al. 2018, Adversarial Examples are not bugs, they are features, Madry et al., 2018)

How to build them



Robustness problem formulation

A trained network $f : \mathcal{D}_x \to \mathcal{D}_y$

Set of input constraint $\mathcal{X} \in \mathcal{D}_x$

Set of output constraint $\mathcal{Y} \in \mathcal{D}_y$

$$\boxed{\text{Verification problem : } x \in \mathcal{X} \Rightarrow f(x) \in \mathcal{Y}}$$

Problem instanciation for adversarial examples

$$\mathcal{X} = \left\{x : \|x - x_0\|_p < \varepsilon\right\}$$
$$\mathcal{Y} = \{y_i : y_i > y_j, \forall j \neq i\}$$

For all perturbations of a sample under a given threshold (*threat model*)
Classification stays unchanged

Limitations and issues

1. Verification problem is usually intractable as it is
2. Adversarial robustness is only relevant to one specific sample (no general characterization for all images)

Enforcing formal robustness for deep learning programs

First approach : testing. . .

Testing suite are a common and useful tool in most of software development to find and get rid of bugs, sometimes automatically.

. . . is not enough

“Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.” (E. Djikstra, 1972).

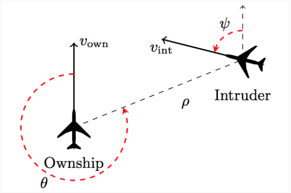
- Remember our goal : have some guarantees on *domains*. Perceptual input spaces is huge and tests cannot cover all possible points.
- Other tools are necessary : formal methods : soundly compute domains of variables to provide mathematical guarantees

Common benchmarks

Adversarial robustness on CIFAR-10 using ConvNets

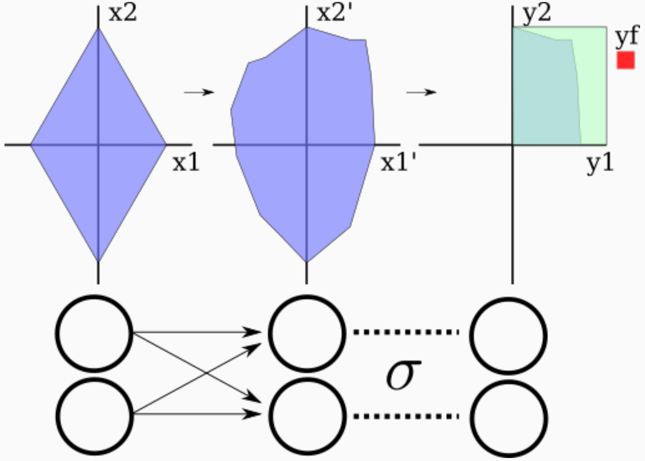
perturbation : l_∞ perturbations with $\varepsilon = 2/255$
metric : *robustness bounds* : how many samples in the test set are certified robust ?

ACAS-Xu



metric : time to check difficult properties
(ϕ_5 and ϕ_{10} from Katz et al., 2017)

Propagation-based algorithms



	DiffAI/DeepZ [1]	CNNCert [2]	Symbolic propagation [3]
Scalability	75s/batch, 16M params	432s/net, 76k params	780s/net, MLP
Completeness	✗	✗	✗
Example results	41% lb	0.0024 certified ε_∞	safe under $\|\|x\|\|_\infty \leq 1$

Table 1 – Recap for propagation-based algorithms

1. Mirman et al., 2018 ; Singh et al., 2019
2. Boopathy et al., 2018
3. Xiang et al., 2017

Optimization/refinement-based algorithms

Base idea : reformulate the problem as an easier optimization problem, compute bounds by solving it

- MILP precompute bounds
- approximated bounds using a dual problem formulation

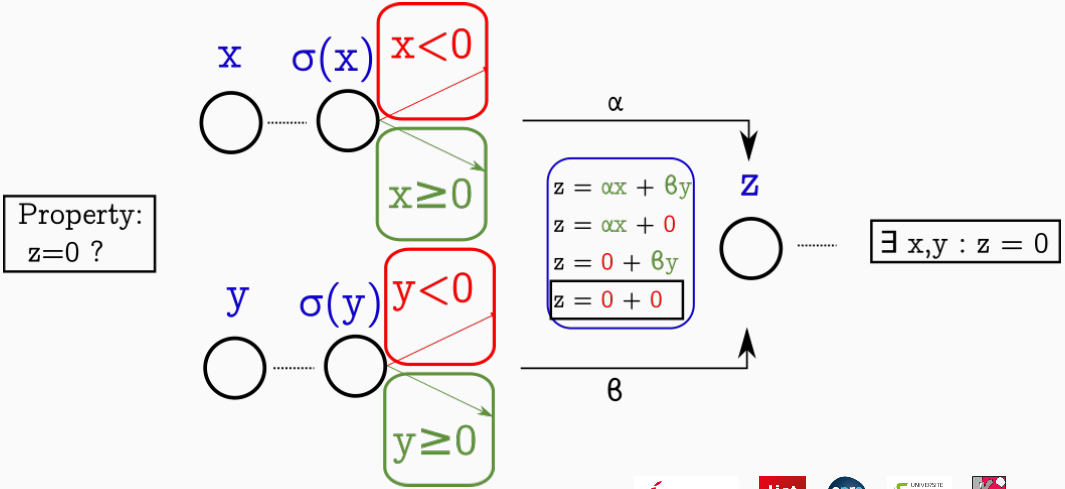
	MILP[4]	Dual problem[5]
Scalability	timed out ACAS ϕ_{10}	proved ACAS ϕ_{10} in 0.003 s
Completeness	✓	✗
Example results	lu 49%, ub 50.2% robustness bounds	ub 53.59% robustness bounds

Table 2 – Recap for refinement-optimization algorithms

4. Tjeng et al., 2017
5. Wong et al., 2017

Search-based algorithms

Base idea : find a counterexample of the property in the search space



Some algorithms

1. ReLuPlex modifies a simplex algorithm to lazily evaluate ReLus
2. Marabou simplifies the network structure
3. ReLUVal search and is guided by symbolic intervals propagation
4. Sherlock uses search using gradient descent augmented with MILP

	ReLuPlex/Marabou [6]	ReLUVal [7]	Sherlock [8]
Scalability	ϕ_5 : 19500s, ϕ_{10} : 2952s	ϕ_5 : 216s	Timed out 24h
Completeness	✓ (Marabou : ✗)	✗	✗
Example results	Sound global robustness properties, safe subspaces identified	Sound global robustness properties, adversarial examples found	Output ranges for control NN

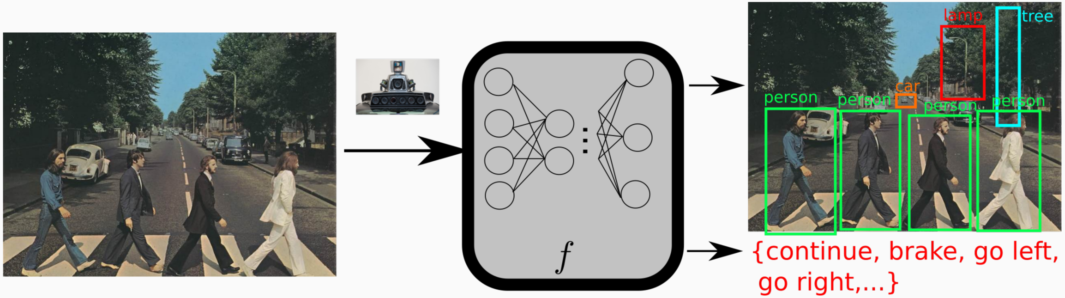
Table 3 – Recap for search-based algorithms

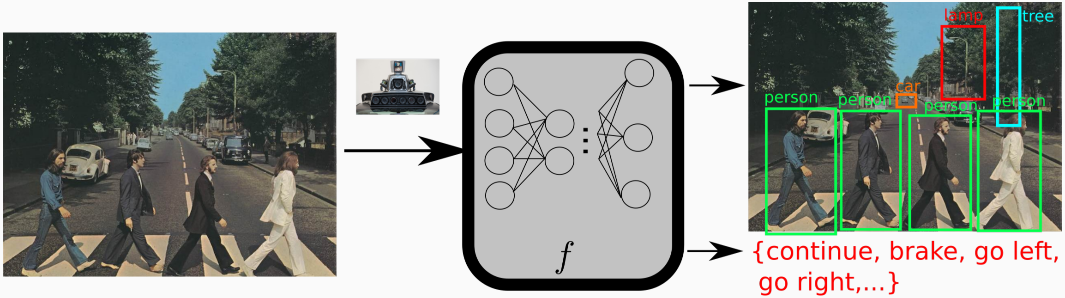
6. Katz et al., 2019
7. Xiang et al., 2018
8. Dutta et al., 2017

Research tracks

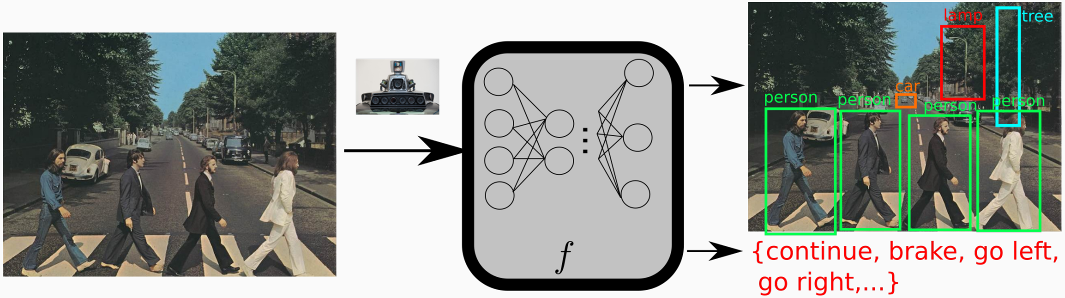
Formally verifying perception

- All adversarial robustness properties are local
- Other work on controllers networks are more global (see Katz. et al.)
- Is there a way to check global properties on perceptual space ?


f
lamp
tree
car
person
person
person
person
{continue, brake, go left, go right,...}

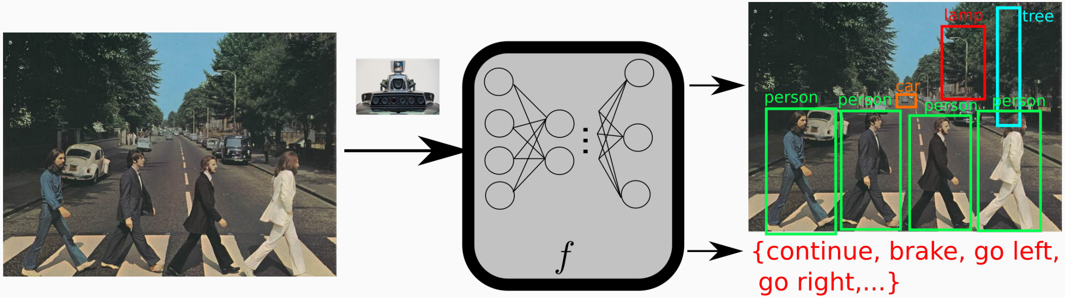


Dream property ϕ : *the autonomous car never run over pedestrians*



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

no *formal characterization* of what a pedestrian is !



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

no *formal characterization* of what a pedestrian is !

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

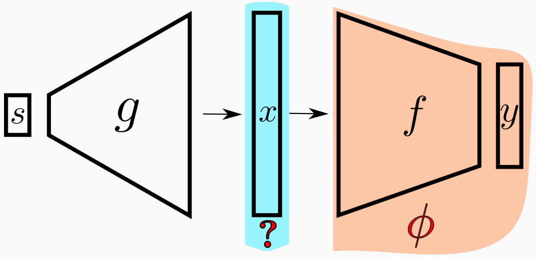
Introducing CAMUS : using simulators to derive a specification

Two main contributions

1. A framwork to express links between simulators and prediction objectives
2. A compiler from ONNX to SMTLIB2

Paper accepted at ECAI 2020 (Girard-Satabin, Julien et al. : *CAMUS : A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators*)

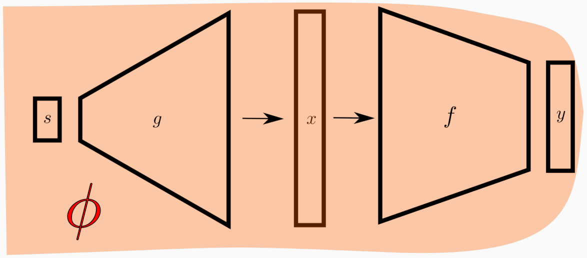
Simulators as data providers



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

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

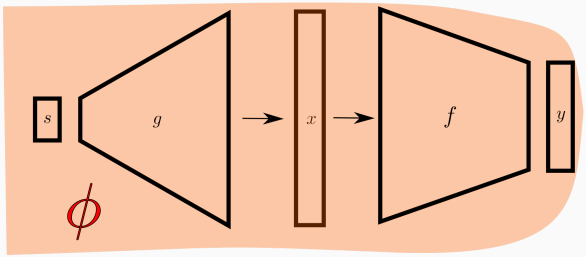
Reformulation of our verification problem



Modify the verification problem formulation to include g and s

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

Reformulation of our verification problem

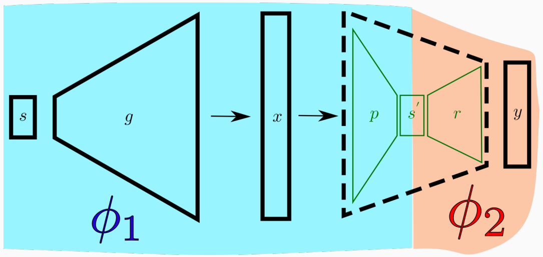


Modify the verification problem formulation to include g and s

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

We now have a property to verify a perceptive unit !

Refinement : splitting perception and reasoning

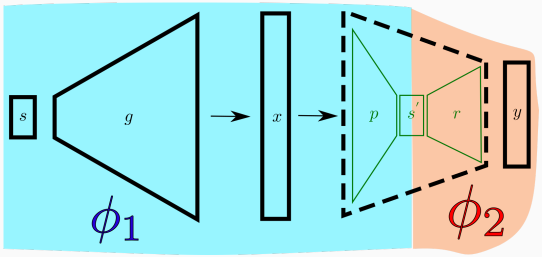


f splits in *p*erception and *r*easoning, p learns s

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 splits in *perception* and *reasoning*, *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)

Express nets under our formalism

Compiler from onnx to logical formulaes (soon open source !)

Future work

1. Sound and complete robustness checking algorithm (scalability is key)
2. Enlarge CAMUS framework to express simulators more efficiently
3. Manage more network architectures and operators
4. Properties expression engine
5. Multiple output targets to improve versatility
6. Others we may not have thought of yet. . .

Final words

Thank you for listening, don't hesitate to shoot your questions :)