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

Enforcing formal robustness for deep learning programs

Research tracks
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
   ⇒ environment modelling difficult and scalability issues
2. Feedforward neural networks are functionally simple (no loops), but variables are meaningless by themselves
   ⇒ current analysis practices not useful
3. Some very specific failure modes, difficult to spot and even more to fix
   ⇒ 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

Formalizing robustness
Enforcing formal robustness for deep learning programs
Research tracks

Formal verification of deep learning
10 février 2020
Robustness problem formulation

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

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

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

Verification problem: $x \in \mathcal{X} \Rightarrow f(x) \in \mathcal{Y}$
Problem instanciation for adversarial examples

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

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.
“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: $l_\infty$ 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 ($\phi_5$ and $\phi_{10}$ from Katz et al., 2017)
Propagation-based algorithms
Formalizing robustness

Enforcing formal robustness for deep learning programs

Research tracks

<table>
<thead>
<tr>
<th>Scalability</th>
<th>DiffAI/DeepZ&lt;sup&gt;1&lt;/sup&gt;</th>
<th>CNNCert&lt;sup&gt;2&lt;/sup&gt;</th>
<th>Symbolic propagation&lt;sup&gt;3&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>75s/batch, 16M params</td>
<td>432s/net, 76k params</td>
<td>780s/net, MLP</td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Example results</td>
<td>41% lb</td>
<td>0.0024 certified $\varepsilon_\infty$</td>
<td>safe under $|x|_\infty \leq 1$</td>
</tr>
</tbody>
</table>

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

Formal verification of deep learning
10 février 2020
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
<table>
<thead>
<tr>
<th>Scalability</th>
<th>MILP$^4$</th>
<th>Dual problem$^5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>timed out ACAS $\phi_{10}$</td>
<td>proved ACAS $\phi_{10}$ in 0.003 s</td>
</tr>
<tr>
<td>Example results</td>
<td>lu 49%, ub 50.2% robustness bounds</td>
<td>x ub 53.59% robustness bounds</td>
</tr>
</tbody>
</table>

**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

Property: $z=0$?

Formal verification of deep learning
10 février 2020
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
<table>
<thead>
<tr>
<th>Research tracks</th>
<th>Scalability</th>
<th>Completeness</th>
<th>Example results</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLuPlex/Marabou$^6$</td>
<td>$\phi_5 : 19500s$, $\phi_{10} : 2952s$</td>
<td>✓ (Marabou : ×)</td>
<td>Sound global robustness properties, safe subspaces identified</td>
</tr>
<tr>
<td>ReLUVal$^7$</td>
<td>$\phi_5 : 216s$</td>
<td>×</td>
<td>Sound global robustness properties, adversarial examples found</td>
</tr>
<tr>
<td>Sherlock$^8$</td>
<td>Timed out 24h</td>
<td>×</td>
<td>Output ranges for control NN</td>
</tr>
</tbody>
</table>

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?
Formalizing robustness
Enforcing formal robustness for deep learning programs

Research tracks

\[ \text{Formal verification of deep learning} \]

10 février 2020
Dream property $\phi$: *the autonomous car never run over pedestrians*
Dream property $\phi$: the autonomous car never run over pedestrians

no *formal characterization* of what a pedestrian is!
Dream property $\phi$: 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 framework to express links between simulators and prediction objectives
2. A compiler from ONNX to SMTLIB2

Simulators as data providers

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

How to formulate $\phi$? What is an image $x$ with a pedestrian?
Modify the verification problem formulation to include $g$ and $s$

$\phi$ 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*
Modify the verification problem formulation to include $g$ and $s$

$\phi$ 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

\[
f \text{ splits in } \text{perception and reasoning, } p \text{ learns } s
\]

\[
\phi_1 \text{ on } p : \text{guarantee of no information loss: reconstruct } s \text{ from } x
\]

\[
s' = s \forall s \rightarrow p \circ g = \text{Id}
\]

\[
\phi_2 \text{ on } r : \text{do not kill pedestrians (assuming perfect perception)}
\]
Refinement: splitting perception and reasoning

\( f \) splits in perception and reasoning, \( p \) learns \( s \)

\( \phi_1 \) on \( p \): guarantee of controlled information loss: \textit{reconstruct} \( s \) from \( x \)
\[ s' \simeq s \quad \forall s \rightarrow p \circ \|g - id\| < \varepsilon \]

\( \phi_2 \) on \( r \): do not kill pedestrians (assuming perfect perception)
Express nets under our formalism

Compiler from onnx to logical formulae (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 :)}