Z3 API in Python, Verification of Neural Nets in Python

Katya Komendantskaya

Pervasive AI...

Pervasive AI...

Autonomous cars

Pervasive AI...

Autonomous cars

Smart Homes

Pervasive AI...

Autonomous cars

Smart Homes

Robotics

Pervasive AI...

Autonomous cars

Smart Homes

Robotics

Chat Bots

Pervasive AI...

Autonomous cars

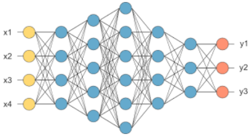
Smart Homes

Robotics

Chat Bots

...and many more ...
AI is in urgent need of verification: safety, security, robustness to changing conditions and adversarial attacks, ...

Neural Nets in Massive use



Used for:

- computer vision
- speech recognition
- (big) data processing
- ...

In:

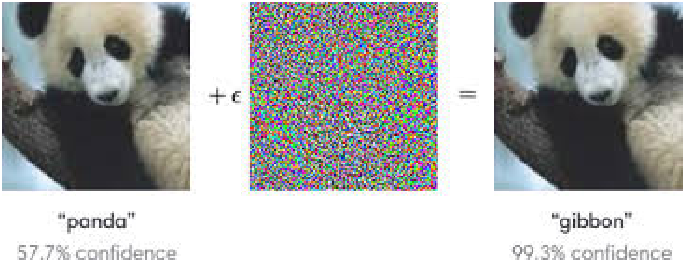
- autonomous cars
- robots
- airport security
- financial applications
- ...
- Alexa
- Google bot on mobile phones
- image recognising apps

Research topics in Neural nets

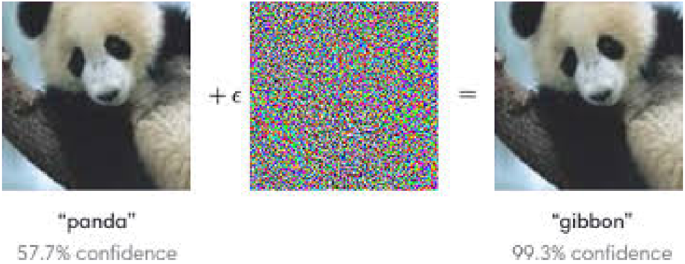
Weaknesses of Neural nets

- not easily conceptualised (lack of explainability)
- prone to error
- prone to adversarial attack

Problems with Neural Net Verification



Problems with Neural Net Verification



- Verification needed: many issues with safety (autonomous devices, cars), security (adversarial attacks)
- Problem: – even to state verification conditions!
- Current methods: Neurons to Logic (*á la* McCulloch and Pitts), Automated Theorem proving, SMT solvers

The literature splits

There are two groups of properties we may want to verify:

- General (concerning properties of learning algorithms): e.g. how well does the learning algorithm perform? do trained neural networks generalise well?

  - A. Bagnall and G. Stewart. Certifying the True Error: Machine Learning in Coq with Verified Generalisation Guarantees. AAAI 2019.
  - A. Bahrami, E. de Maria and A.Felty. Modelling and Verifying Dynamic Properties of Biological Neural Networks in Coq. 2018.

The literature splits

There are two groups of properties we may want to verify:

- General (concerning properties of learning algorithms): e.g. how well does the learning algorithm perform? do trained neural networks generalise well?

  A. Bagnall and G. Stewart. Certifying the True Error: Machine Learning in Coq with Verified Generalisation Guarantees. AAAI 2019.

  A. Bahrami, E. de Maria and A.Felty. Modelling and Verifying Dynamic Properties of Biological Neural Networks in Coq. 2018.

- Specific to applications (concerning neural network deployment): given this trained neural network, is it robust to adversarial attacks?

  X. Huang and M. Kwiatkowska and S. Wang and M. Wu. Safety Verification of Deep Neural Networks. CAV (1) 2017: 3-29

  G. Singh, T. Gehr, M. Puschel, M. T. Vechev: An abstract domain for certifying neural networks. PACMPL 3(POPL): 41:1-41:30 (2019)

Robustness Verification scenario

SMT Solvers

- Widely used in verification

Robustness Verification scenario

SMT Solvers

- Widely used in verification
- Z3 is one of the most popular

Robustness Verification scenario

SMT Solvers

- Widely used in verification
- Z3 is one of the most popular
- There is Z3 API in Python

Robustness Verification scenario

SMT Solvers

- Widely used in verification
- Z3 is one of the most popular
- There is Z3 API in Python
- It is often used in NN verification

Robustness Verification scenario

SMT Solvers

- Widely used in verification
- Z3 is one of the most popular
- There is Z3 API in Python
- It is often used in NN verification

Example of Z3 Python API at work:

```
from z3 import *

x = Int('x')
y = Int('y')
solve(x > 2, y < 10, x + 2*y == 7)
```

Robustness Verification scenario

SMT Solvers

- Widely used in verification
- Z3 is one of the most popular
- There is Z3 API in Python
- It is often used in NN verification

Example of Z3 Python API at work:

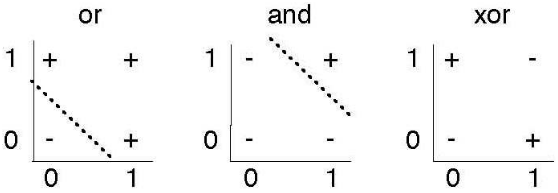
```
from z3 import *

x = Int('x')
y = Int('y')
solve(x > 2, y < 10, x + 2*y == 7)
```

- will find all solutions or say that no exist.
- DEMO of Z3 in Python

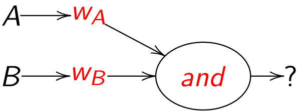
Simple example: Perceptron

Neural nets doing logic [McCulloch and Pitts, 1943]:



A	B	A and B	A or B	A xor B
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

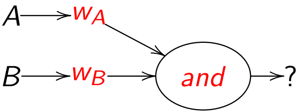
Perceptron for and



Input features and target features:

A	B	A and B
true	true	true
true	false	false
false	true	false
false	false	false

Perceptron for and

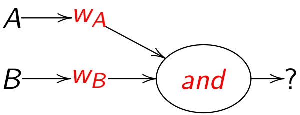


Input features and target features:

A	B	A and B
true	true	true
true	false	false
false	true	false
false	false	false

Now train the network: will it be able to learn the correct (linear) function $\theta + w_A \times A + w_B \times B$ to simulate and?

Perceptron for and



Input features and target features:

A	B	A and B
true	true	true
true	false	false
false	true	false
false	false	false

Now train the network: will it be able to learn the correct (linear) function $\theta + w_A \times A + w_B \times B$ to simulate and?
e.g. $-0{,}9 + 0{,}5 \times A + 0{,}5 \times B$

General motivation

General motivation is to make the solver:

- solve constraints like:

```
(forall m n. ( (m >=. 0.5R) /\ (n >=. 0.5R)) ==>
     (nextoutput perceptron [m ; n]) == 1)
```

General motivation

General motivation is to make the solver:

- solve constraints like:

```
(forall m n. ( (m >=. 0.5R) /\ (n >=. 0.5R)) ==>
    (nextoutput perceptron [m ; n]) == 1)
```

- ... that is, to guarantee correctness of inputs for certain small range of deviations from the well-classified input

General motivation

General motivation is to make the solver:

- solve constraints like:

```
(forall m n. ( (m >=. 0.5R) /\ (n >=. 0.5R)) ==>
    (nextoutput perceptron [m ; n]) == 1)
```

- ... that is, to guarantee correctness of inputs for certain small range of deviations from the well-classified input
- If the new input does not fall within the set constraints, it cannot be guaranteed to be robust to attack

General motivation

General motivation is to make the solver:

- solve constraints like:

```
(forall m n. ( (m >=. 0.5R) /\ (n >=. 0.5R)) ==>
    (nextoutput perceptron [m ; n]) == 1)
```

- ... that is, to guarantee correctness of inputs for certain small range of deviations from the well-classified input
- If the new input does not fall within the set constraints, it cannot be guaranteed to be robust to attack
- Imagine an autonomous car passing control to a human if it cannot guarantee robust recognition of some crucial street signs.

Simple Robustness Verification scenario

- Implement my Perceptron in Python

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:
  - Define its robustness region: e.g. when input array contains real values in the region ϵ given (say) by some Eucledian distance from our ideal input $[1, 1]$

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:
  - Define its robustness region: e.g. when input array contains real values in the region ϵ given (say) by some Eucledian distance from our ideal input $[1, 1]$
  - define a step function ("the ladder") to generate a finite number of reals in this region (otherwise Z3 will not terminate)

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:
  - Define its robustness region: e.g. when input array contains real values in the region ϵ given (say) by some Eucledian distance from our ideal input $[1, 1]$
  - define a step function ("the ladder") to generate a finite number of reals in this region (otherwise Z3 will not terminate)
  - Prove the ladder is "covering" (using pen and paper)

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:
  - Define its robustness region: e.g. when input array contains real values in the region ϵ given (say) by some Eucledian distance from our ideal input $[1, 1]$
  - define a step function ("the ladder") to generate a finite number of reals in this region (otherwise Z3 will not terminate)
  - Prove the ladder is "covering" (using pen and paper)
  - Take the set of input generated by Z3, run them through the Perceptron

Simple Robustness Verification scenario

- Implement my Perceptron in Python
- Prove it is robust for class 1:
  - Define its robustness region: e.g. when input array contains real values in the region ϵ given (say) by some Eucledian distance from our ideal input $[1, 1]$
  - define a step function ("the ladder") to generate a finite number of reals in this region (otherwise Z3 will not terminate)
  - Prove the ladder is "covering" (using pen and paper)
  - Take the set of input generated by Z3, run them through the Perceptron
  - No mis-classification? – I have proven my network robust for output 1, region ϵ and the ladder.

Demo

... of how this methodology is reaslised in Python with Z3.