Scaling Polyhedral Neural Network Verification on GPUs

François Serre, Christoph Müller*, Gagandeep Singh, Markus Püschel and Martin Vechev*

fserre.github.io

Department of Computer Science ETH zürich

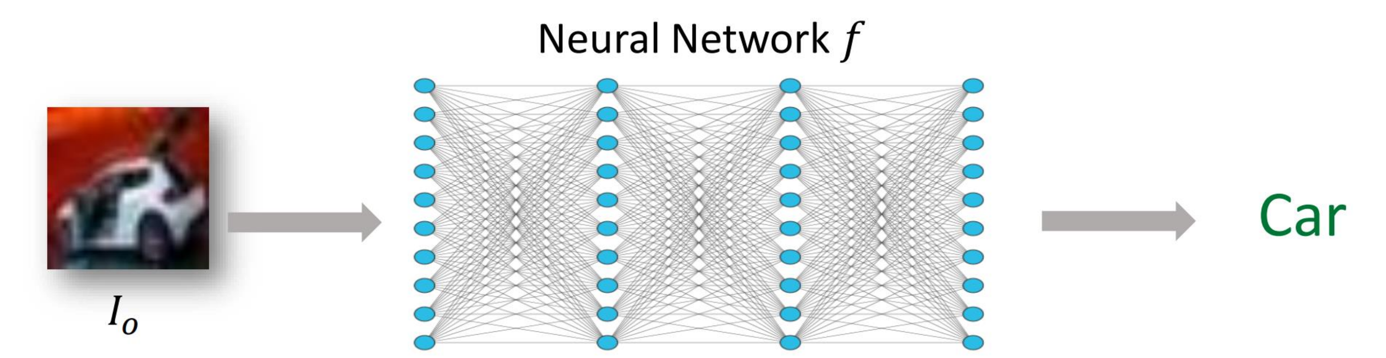
vmware Research

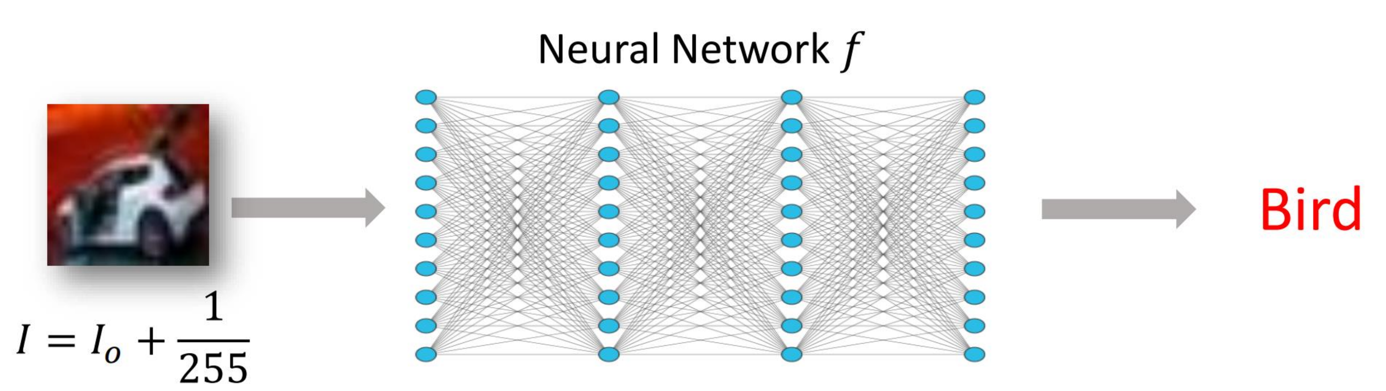
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Verifying the robustness of deep neural networks

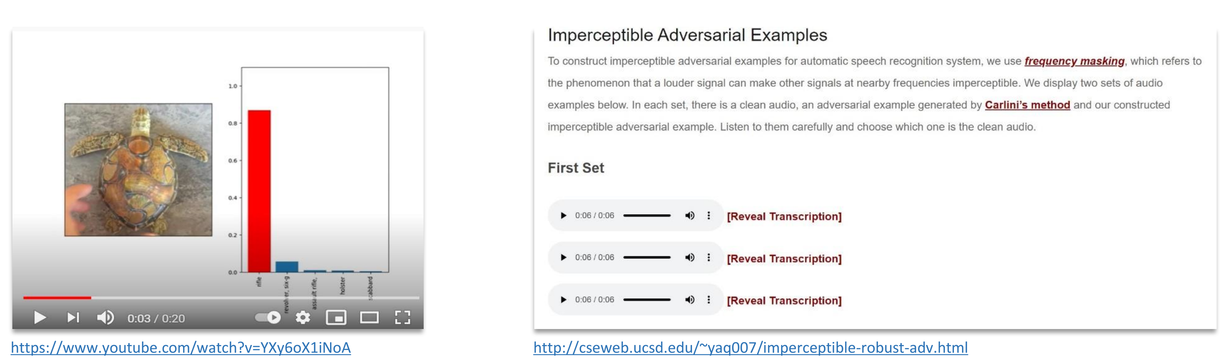
Context: adversarial attacks on DNN

Deep neural networks are vulnerable to adverserial examples:





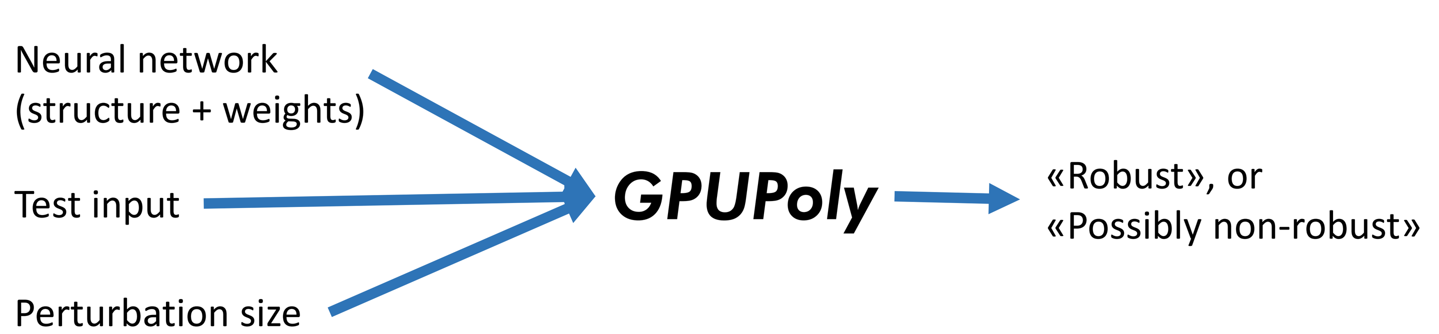
There is an active research on robust and targetted and attacks:



We need a way to verify the robustness of a network to such attacks.

Neural network robustness verifier

Exact neural network certification is NP-hard. We need to overapproximate:



Our approach is applicable beyond intensity based robustness verification of image classifiers.

Source code

https://github.com/eth-sri/ELINA/tree/master/gpupoly

References

An Abstract Domain for Certifying Neural Networks.
Gagandeep Singh, Timon Gehr, Markus Püschel and Martin Vechev
Proc. ACM Program. Lang., 2019, pp 41:1-41:30

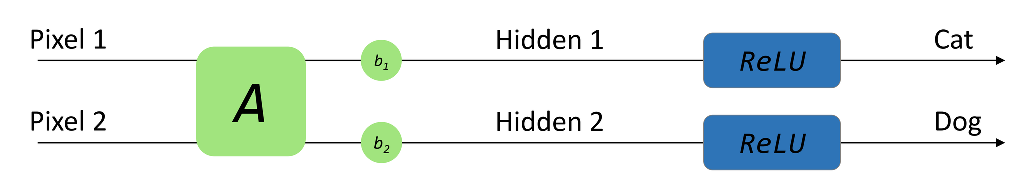
Towards Stable and Efficient Training of Verifiably Robust Neural Network
H, Zhang, H, Chen, C, Xiao, S, Gowal, R, Stangorth, B, Li, D, Boning, C, Hsieh
Proc. International Conference on Learning Representations, 2020

** Equal contribution*

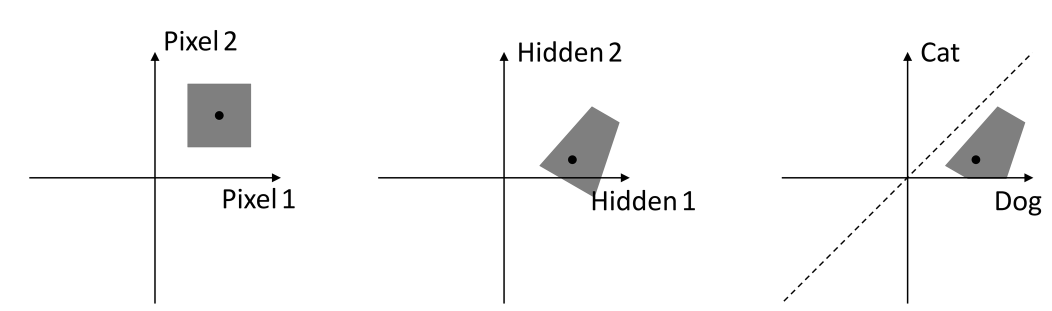
Combining two models for speed and accuracy

Example on a small neural network

We consider a two-input binary classifier:



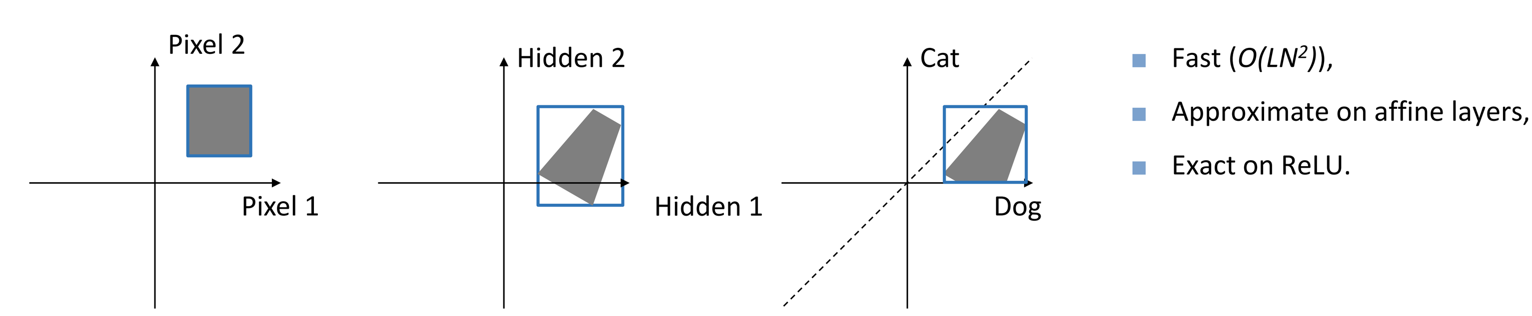
In this (small) case, the response of this classifier on the neighborhood (in grey) of an input (black point) can be computed exactly:



GPUPoly combines the two following relaxations:

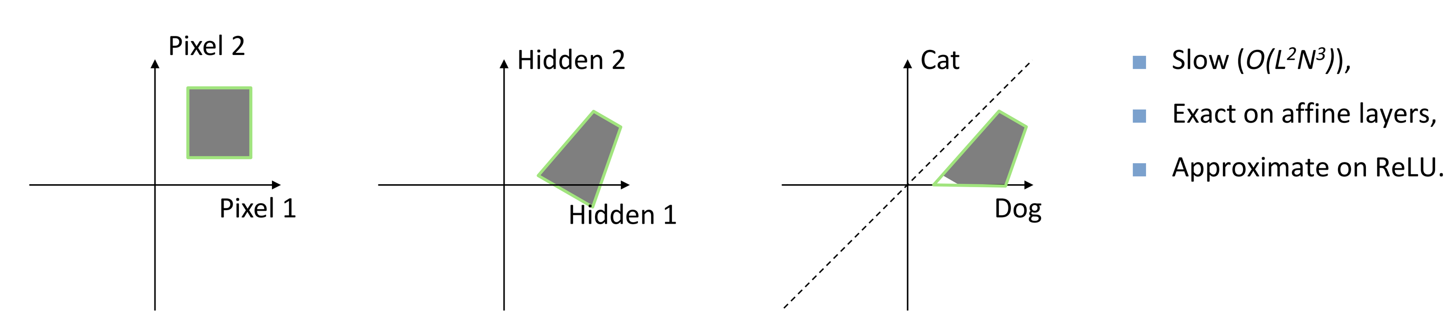
Bounding box model and forward interval analysis

Forward interval analysis is one of the fastest, at the price of precision:



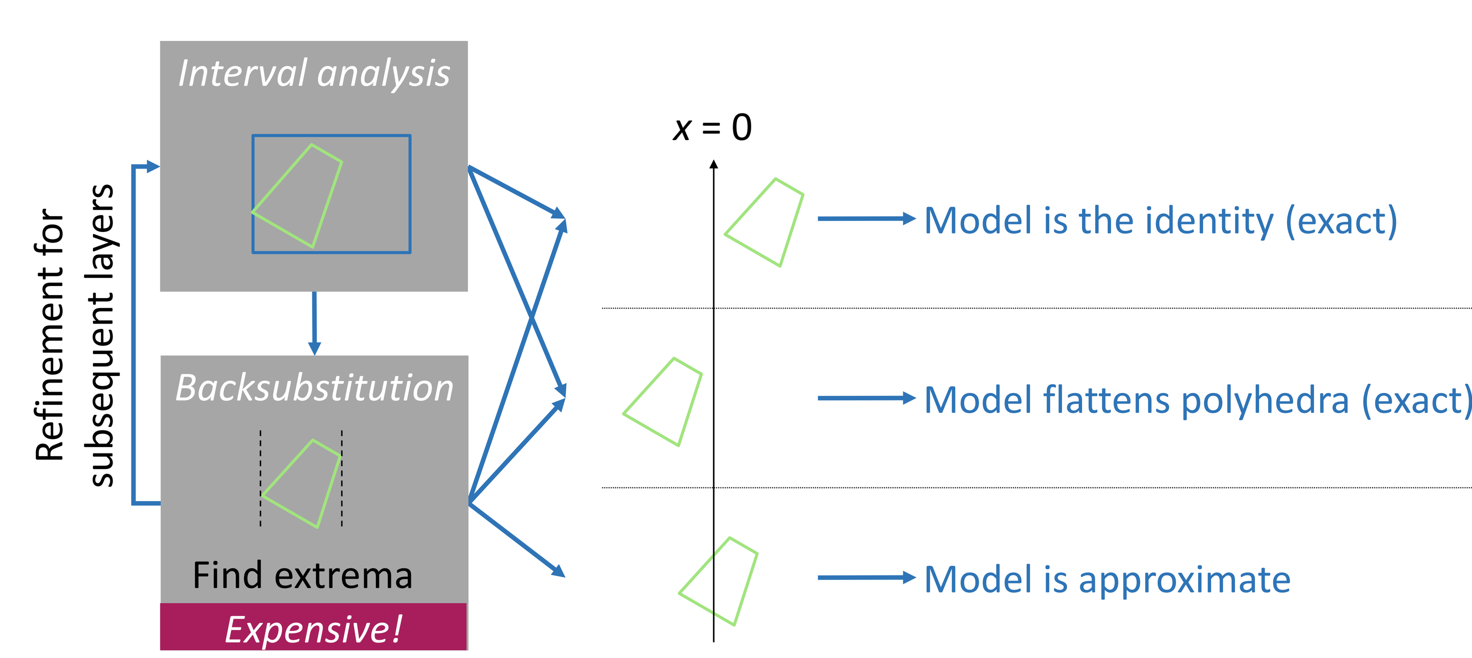
Polyhedron model and DeepPoly analysis [POPL'19]

DeepPoly is more precise, but slower:



GPUPoly: an hybrid

GPUPoly uses DeepPoly relaxation, but tries to skip *backsubstitution*, its bottleneck, using forward interval analysis:



GPU implementation and results

Implementation of GPUPoly

GPUPoly is implemented to run on GPU, as it uses extensively embarassingly parallelizable linear algebra functions.

These functions are custom-made, for compliance with *floating-point soundness*, and to exploit sparsity patterns in case of successive convolutional layers.

Floating-point (FP) soundness

GPUPoly yields certifications that take into account rounding errors that may occur during its own computations, and during inference. Namely,

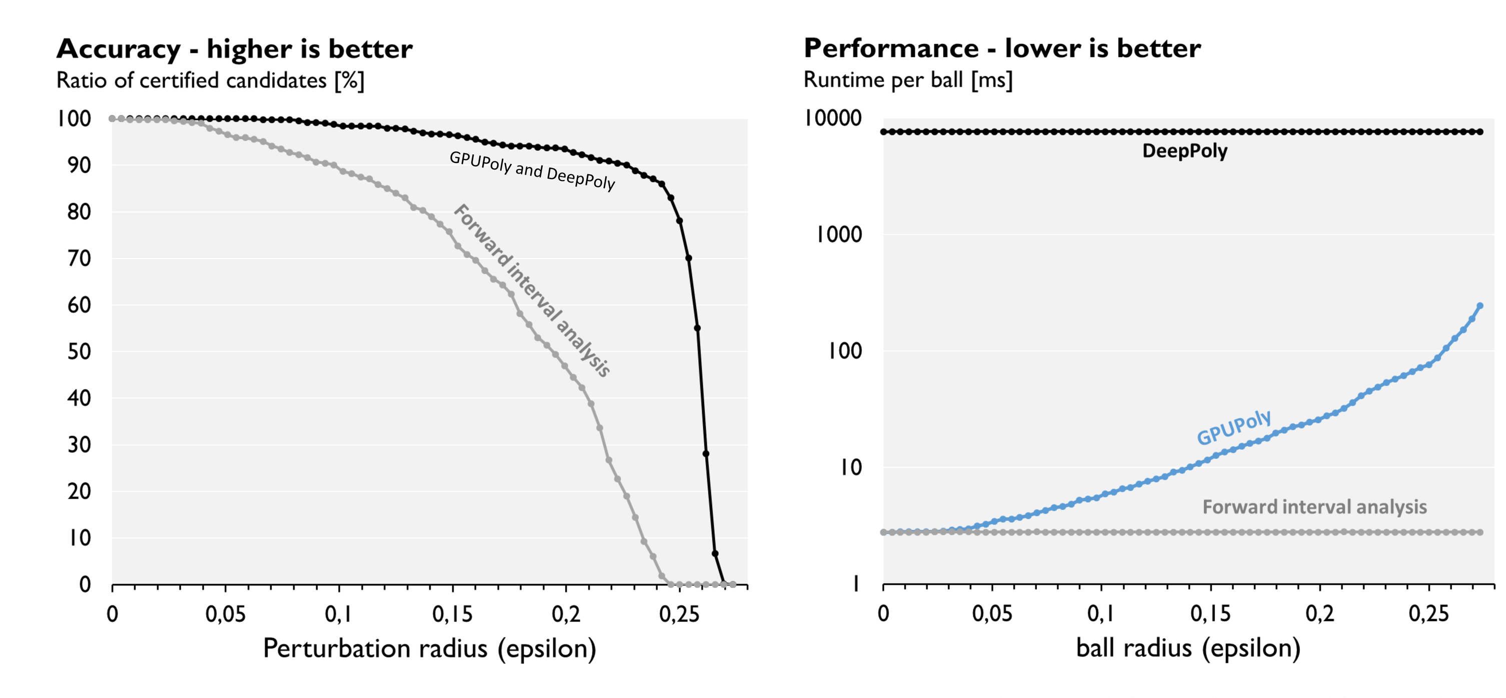
- FP arithmetic has finite precision: 1 and 2^{-24} are both representable, but not their sum.

  Need to use directed rounding, and interval arithmetic.

- FP addition is not associatative: the value of $2^{-24} + 1 - 1$ depends on evaluation order.

  Need to use the next representable number for summands.

Comparison with Forward interval Analysis and DeepPoly [POPL'19]



GPUPoly is as precise as DeepPoly, while being significantly faster on most networks. A notable exception are networks trained with projected gradient descend, for which it has the same runtime as DeepPoly.

Comparison with Crown-IBP [ICLR'20]

Model	#Neurons	Training	ϵ	#Candidates	#Verified		Median runtime	
					CR-IBP	GPUPoly	CR-IBP	GPUPoly
MNIST Dataset, 10k test inputs								
6×500	3,010	Normal	8/255	9,844	0	7,291	130 µs	9.06 ms
ConvBig	48K	DiffAI	3/10	9,703	5,312	8,809	220 µs	537 µs
ConvSuper	88K	Normal	8/255	9,901	0	8,885	300 µs	266 ms
IBP_large_0.2	176K	CR-IBP	0.258	9,895	4,071	7,122	190 µs	9.04 ms
IBP_large_0.4	176K	CR-IBP	3/10	9,820	9,332	9,338	190 µs	2.92 ms
CIFAR 10 Dataset, 10k test inputs								
6×500	3,010	Normal	1/500	5,607	0	4,519	200 µs	8.04 ms
ConvBig	62K	DiffAI	8/255	4,599	1,654	2,650	320 µs	730 µs
ConvLarge	230K	DiffAI	8/255	4,615	1,672	2,838	900 µs	4.54 ms
IBP_large_2_255	230K	CR-IBP	2/255	7,082	5,450	5,588	820 µs	12.3 ms
IBP_large_8_255	230K	CR-IBP	8/255	4,540	3,289	3,298	270 µs	3.83 ms
CIFAR 10 Dataset, first 1k test inputs								
ResNetTiny	311K	PGD	1/500	768	0	651	2.76 ms	11.7 s
ResNet18	558K	PGD	1/500	823	0	648	7.13 ms	397 s
ResNetTiny	311K	DiffAI	8/255	371	217	244	2.93 ms	4.03 ms
SkipNet18	558K	DiffAI	8/255	321	245	260	7.29 ms	16.5 ms
ResNet18	558K	DiffAI	8/255	372	238	268	7.49 ms	16.9 ms
ResNet34	967K	DiffAI	8/255	356	200	229	13.8 ms	34.5 ms