Imperial College London

Multi-Dimensional Domain Splitting for Neural Network Verification

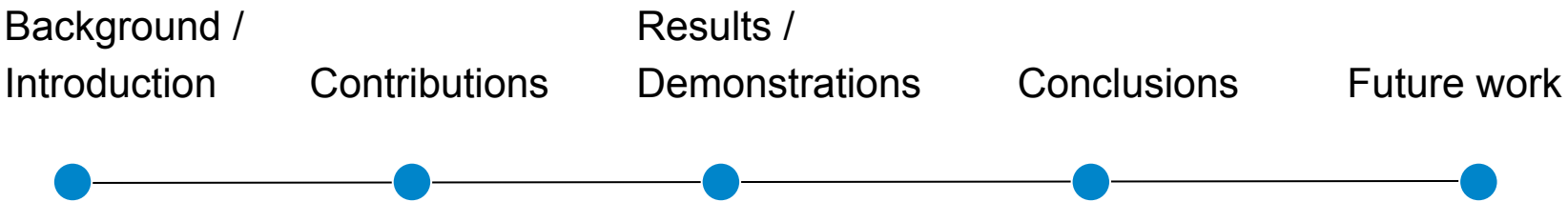
Msc Project Presentation

James Dorricott

11 September 2020

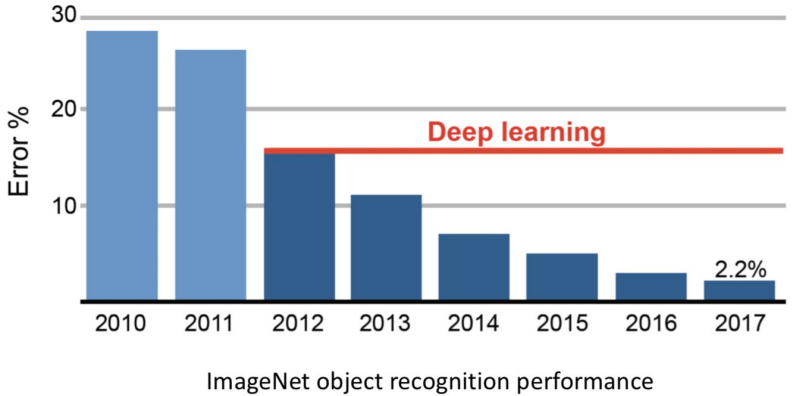
Imperial College London

What I will cover



Imperial College London

Explosion in deep learning...



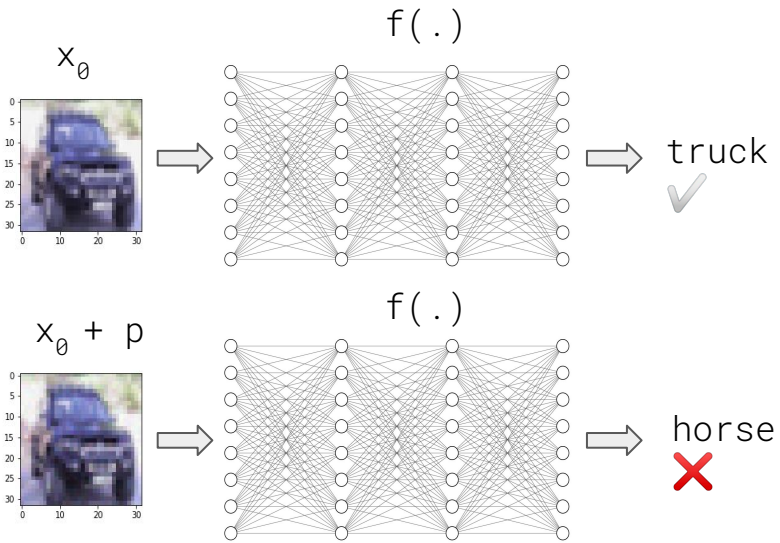
ImageNet object recognition performance

Figures: M. Bronstein

Imperial College London

Neural network errors

- It's well-known that neural networks are vulnerable to *adversarial* images
- These are images that look the same to a human but lead to an incorrect classification
- These need to be identified before networks are deployed in safety-paramount applications



Imperial College London

Neural network verification

- The field of neural network verification deals with this *mathematically*
- To make it meaningful requires pre- and post-conditions on the network
- Showing that these hold is the “open-loop verification problem”
- Solving this problem on high-dimensional inputs is the core focus of this project

‘Preconditions’	=	*Input assumptions*
‘Postconditions’	=	*Output properties*
Neural Network	=	*Trained weights/biases*

Imperial College London

Terminology: the verification problem

- We can define this more formally...
- We use the infinity-norm the capture all possible input perturbations up to some ε-bound
- This problem allows for an exact representation as a **MILP program**[1]

Definition 1 *Verification problem. Let* $\mathcal{X}_0 \subseteq \mathbb{R}^m$ *and* $\mathcal{Y} \subseteq \mathbb{R}^n$ *be input and output constraints, respectively, imposed on a neural network network* $f : \mathbb{R}^m \to \mathbb{R}^n$. *Then, the verification problem is to establish whether for all* $\mathbf{x} \in \mathcal{X}_0$, $\mathbf{y} \in \mathcal{Y}$.

Definition 2 *Local robustness. Local robustness is a pair* $(\mathcal{X}_0, K)$, *where* $\mathcal{X}_0 \subseteq \mathbb{R}^m$ *defines a bounded input region and* $1 \leq K \leq n$ *represents the index of the desired class for all inputs in* $\mathcal{X}_0$. *A neural network is said to be locally robust if for all* $\mathbf{x} \in \mathcal{X}_0$, *we have that* $(f(\mathbf{x}))_K > (f(\mathbf{x}))_j$ *for all* $j \neq K$.

1: A. Lomuscio and L. Maganti, “An approach to reachability analysis for feed-forward relu neural networks,” 2017

Imperial College London

Terminology: MILP encodings

- To do so requires a way to linearly encode the non-linearity of the network
- The use of a Big-M encoding[1] can precisely capture the possible phases of the ReLU when the pre-activation pattern is unknown
- The encoding we use leverages the pre-activation **bounds** and modifies slightly the original formulation of Lomuscio & Maganti (2017)

$$
\begin{aligned}
z_{i,j} &\geq \hat{z}_{i,j} \\
z_{i,j} &\geq 0 \\
z_{i,j} &\leq \hat{u}_{i,j}\delta_{i,j} \\
z_{i,j} &\leq \hat{z}_{i,j} - \hat{l}_{i,j}(1-\delta_{i,j})
\end{aligned}
$$

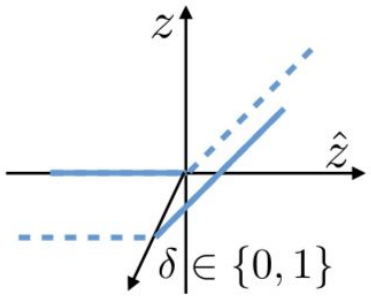


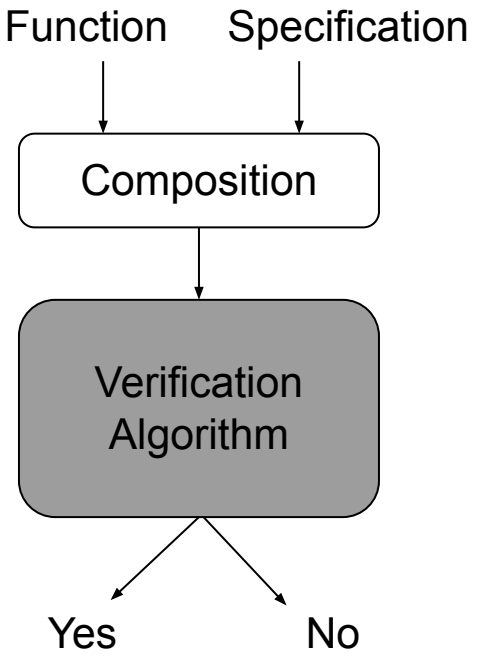
Figure: C. Liu

2: V. Tjeng, K. Xiao, and R. Tedrake, “Evaluating robustness of neural networks with mixed integer programming,” 2017

Imperial College London

Verification pipeline: template

1. Define the input constraints
2. Define the output constraints
3. Starting from the input, propagate the bounds[3] throughout the network
4. Encode unstable nodes as MILP constraints
5. Dispatch all constraints to a MILP solver
6. If the **answer** to the satisfiability problem is NO, the network is proven to be robust within the given input region



3: We use the symbolic method for computing the bounds of Wang et al (2018)

Imperial College London

Domain splitting: bisection

- Tighter bounds still can be found by *bisecting* the input domain
- Bisection proceeds by dividing the input in half **one dimension** at a time
- Bisection has been shown to be effective for low-dimensional inputs[4,5]
- Different methods exist for how to optimise to dimension to split at each step

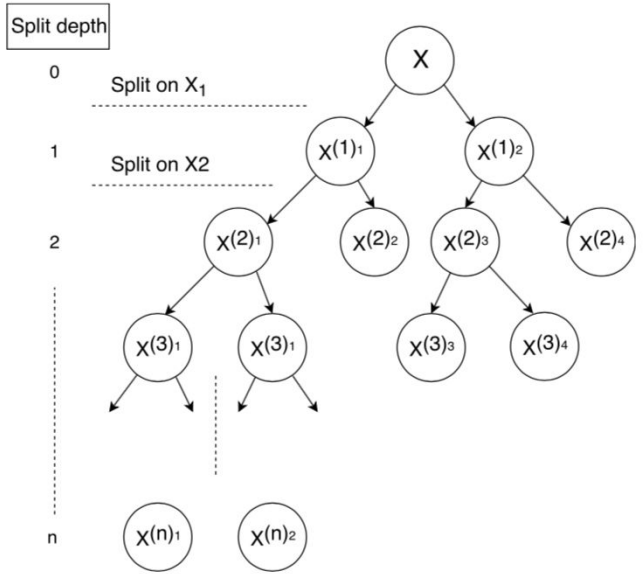


Figure: S. Wang

4: S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, “Formal security analysis of neural networks using symbolic intervals,” 2018
5: R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, “A unified view of piecewise linear neural network verification,” 2018

Imperial College London

Multi-dimensional splitting: motivation

- The success of efficiently reaching an answer[6] to the verification problem when formulated with MILP constraints is related to:
  - The number of binary variables ≡ the number of unstable ReLUs
  - The strength of the relaxations ≡ the tightness of the pre-activation bounds
- This means that networks with high-dimensional inputs (often with many hidden nodes as well) can be very challenging to verify
- And bisection is not well-suited to high-dimensional inputs:
  - One split will be unlikely to reduce the overall problem complexity
  - Exponential blow-up of subregions if the splitting depth is not limited
  - Forced to randomly select which dimension to split

6: Modern commercial solvers apply the *branch and bound* algorithm for solving MILP problems

Imperial College London

Summary of contributions

- *In this project, we…*
- Present a new algorithm[7] for splitting the input domain along **multiple dimensions** which fixes a given ReLU node into the inactive or active state
- Perform extensive investigations to identify the best splitting strategies
- Engineer a distributed verification system for sharing jobs over a network
- Investigate the use of hybrid MILP formulations for checking a combination of the original verification problem and our new splitting procedure

7: Proposed by E. Botoeva (2020)

Imperial College London

The splitting algorithm

- Assume the input region is the multi-dimensional interval [**l**, **u**] and that the equation e(**x**) = **wx** + b estimates the upper bound of a given node y with input **x** ∈ R^m
- Finding split locations $s_{i1},...,s_{ik}$ in dimensions i,...,k that fix y into the inactive state can be solved by means of a MILP

Definition 3 *Splitting algorithm encoded as a MILP.*

- *Take m binary variables $\delta_i, \ldots, \delta_m$, with the meaning that $\delta_i = 1$ iff we split dimension i.*
- *Take m real variables $s_i, \ldots, s_m$, such that when $\delta_i = 1$, the value of $s_i \in [l_i, u_i]$ is the location of the split of input dimension i.*
- *Take m real variables $u_i^*, \ldots, u_m^*$, such that when $\delta_i = 1$, $u_i^* = s_i$, and when $\delta_i = 0$, u_i^* is equal to u_i or l_i depending on $sgn(w_i)$.*

The following constraints are added to the program:

- $e(\boldsymbol{u}^*) = 0$, *where* $\boldsymbol{u}^* = [u_1^*, \ldots, u_m^*]^T$.
- $\delta_i + \cdots + \delta_m \geq 1$.

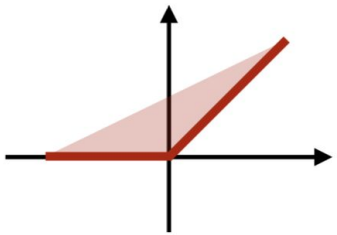
We also add the following objective function:

- *Minimise the sum* $\delta_i + \cdots + \delta_m$.

Imperial College London

Splitting heuristics

- To decide which node to fix in the network, we make use of the triangle relaxation[8] as a measure of a node's *instability*:
  - The smaller the value of the area induced by the relaxation, the closer the node is to being stable



$\text{Area}_{\text{relax}} := \frac{1}{2}(-l)(u)$

Example: Picking the node with the *2*nd smallest over-approximation area

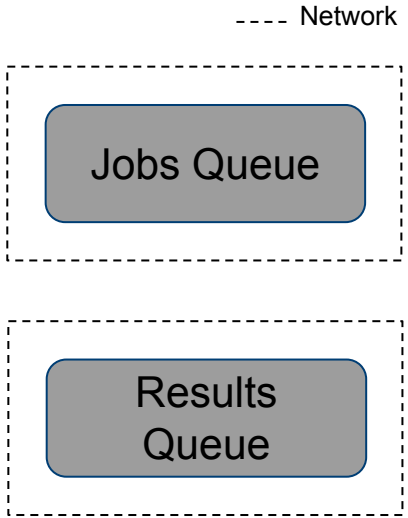
Node	$\text{Area}_{\text{relax}}$
(1,2)	0.8
(2,3)	1.6
(2,1)	3.4
⋮	⋮

8: R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," 2017

Imperial College London

Distributed verification

- Because a splitting step creates a number of independent problems on each of the resulting subregions, we can check these in *parallel*
- A core contribution of this project: building on the existing parallelization scheme[9] by allowing verification jobs to be shared over a network of machines
- To achieve this, we:
  - Leveraged the server/client manager framework available in the Python std. library and the concept of *proxy* objects
  - Designed the system to allow for fully automatic connections

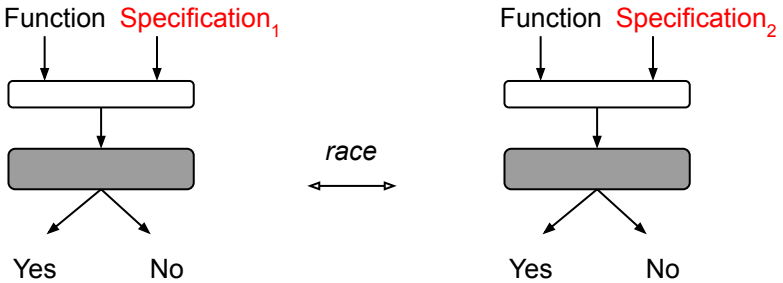


9: Included as part of the inherited verification toolkit for this project

Imperial College London

Hybrid MILP formulations

- *Infeasible* instances (solver returns NO) can often be verified more quickly using the standard MILP encoding
  - ...whereas (see next slide) our splitting procedure is good at speeding up the detection of counter-examples (solver returns YES)
- We have therefore investigated the use of hybrid/combined formulations that enter into a parallel race against each other to produce the answer



Example: checking the original problem (‘*spec 1*’) in parallel to our splitting procedure (‘*spec 2*’)

Imperial College London

Headline result

- We performed extensive experiments[10] on a five-layer CIFAR10 classifier with 3,072 inputs / 2,048 ReLUs
- Key finding: *our splitting procedure is able to speed up (x2) counter-example detection on networks with high-dimensional inputs*

Splitter	Queries solved	Total time (s)	Avg time (solved) (s)
None	84	44,309.51	184.62
Bisection	90	32,695.49	163.28
Multi-dim	93	23,733.63	119.71

Table 1: The best verification result of our multi-dimensional splitter applied to 100 mixed-difficulty, non-robust CIFAR instances at ε = 0.03. The timeout was set to 1,800 seconds. We compare it against the original problem (no domain splitting) and a bisection splitter.

10: We also observed good results (x3 speedup) on a smaller-sized MNIST network (784 inputs, 1,024 hidden nodes). We also extended our toolkit to work with the ACAS Xu safety networks, but initial experiments were inconclusive

Imperial College London

Additional insights

- Fixing unstable nodes via domain splitting serves to (a) reduce to search space for the MILP solver and (b) improve the strength of the LP relaxations
- Rather than waiting until a pre-specified stability ratio, e.g., 70%, has been reached (which may require many splitting steps), we found our most effective splitting strategy was to:
  - Perform one splitting step that can immediately produce subproblems for verification
  - Fix the *n*th almost-stable node as a *heuristic* for fixing all nodes with smaller $\text{Area}_{\text{relax}}$
  - Ensure the method for choosing *n* is such that *n* remains **small**
  - ...which suggests that it is only necessary to fix a few unstable nodes to significantly improve upon the original formulation (“dependency effect”)

Imperial College London

Demo: distributed system

Example: the server/client connections and readout from the remote workers are shown in real-time

Imperial College London

Hybrid investigation: results

- In the report, we demonstrate the correctness of our distributed system
- We also used it to investigate our hypothesis that if robustness is not known *a priori*, it's logical to check the original problem and our splitting procedure

Approach	Queries solved	Total time (s)
Baseline	83	54,249.04
Combination	88	37,766.93

Table 2: The result of our combined approach applied to 100 mixed-difficulty, robust AND non-robust CIFAR instances at ε = 0.03. The timeout was set to 1,800 seconds. The combined approach was performed on our distributed system with 2 client verifiers. We compare it against the original problem (no domain splitting).

Imperial College London

Conclusions

- We have introduced a new, multi-dimensional splitting algorithm that is able to more quickly detect counter-examples and brings some key advantages:
  - Based on exploiting information readily available, namely, ReLU stability
  - Aimed directly at reducing the number of binary variables to consider
  - Unlike bisection (which can't be optimised when the dimensionality is high), minimal overhead means it should be able to scale up to larger networks than tested in this project
- We have built a first prototype of a distributed verification system that is able to share jobs over a network, with positive initial demonstrations
- We have investigated the use of combined MILP formulations as an interesting solution to verification *in the wild*

Imperial College London

Future work

- Scaling verification algorithms, particularly complete methods, to real-world networks remains a key challenge. Does our approach hold promise here?
  - ...or do more hidden nodes simply imply more nodes need to be fixed in a splitting step, thus increasing the overhead from our splitting procedure?
  - This is a key trade-off to be considered
- How can we build a more scalable distributed system with less memory cost?
  - May require workarounds, or more production-grade tools than the Python std. library
- Are they other ways of approaching combined formulations?
  - A different approach[11] is to first check the original problem and then only split the input domain when a pre-specified timeout has been reached

11: H. Wu, A. Ozdemir, A. Zeljic, A. Irfan, K. Julian, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, and C. Barrett, “Parallelization techniques for verifying neural networks,” 2020

Imperial College London

Thank you!