Design, Test, Repair: Software Quality for Neural Networks

ESSAI 2025

Julien Girard-Satabin
Zakaria Chihani
Dorin Doncenco

CEA LIST

2025-07-03

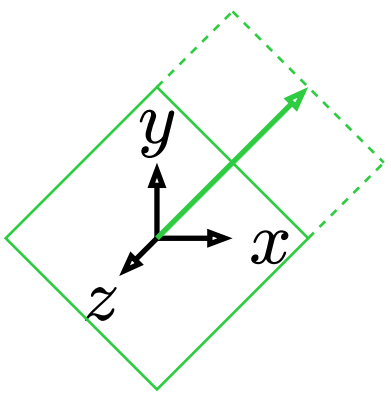
This work was supported by the French Agence Nationale de la Recherche (ANR) through SAIF (ANR-23-PEIA-0006) and DeepGreen (ANR-23-DEGR-0001) as part of the France 2030 programme.

Recaps on previous course

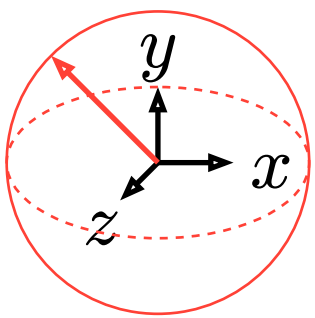
Local robustness (Katz et al. 2017)

Let a classifier $f : \mathcal{X} \mapsto \mathcal{Y}$. Given $x \in \mathcal{X}$ and $\varepsilon \in \mathbb{R} \lll 1$ the problem of *local robustness* is to prove that $\forall x'.\ \|x - x'\|_p < \varepsilon \rightarrow f(x) = f(x')$

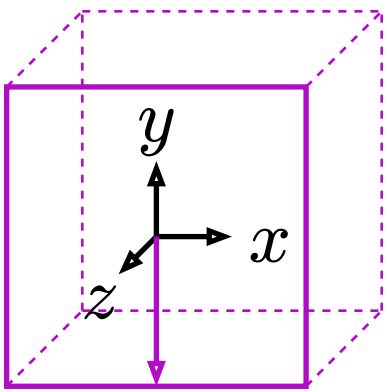
$$L_1(x) = \sum_i |x_i|$$



$$L_2(x) = \sqrt{\sum_i (x_i)^2}$$



$$L_\infty(x) = \max_i |x_i|$$



This session

1. Testing neural networks with logical relations
2. Bugfixing neural networks, with guarantees
3. Encoding logical constraints within neural networks
4. Open questions on properly evaluating the « quality » of ML programs

A brief introduction on testing

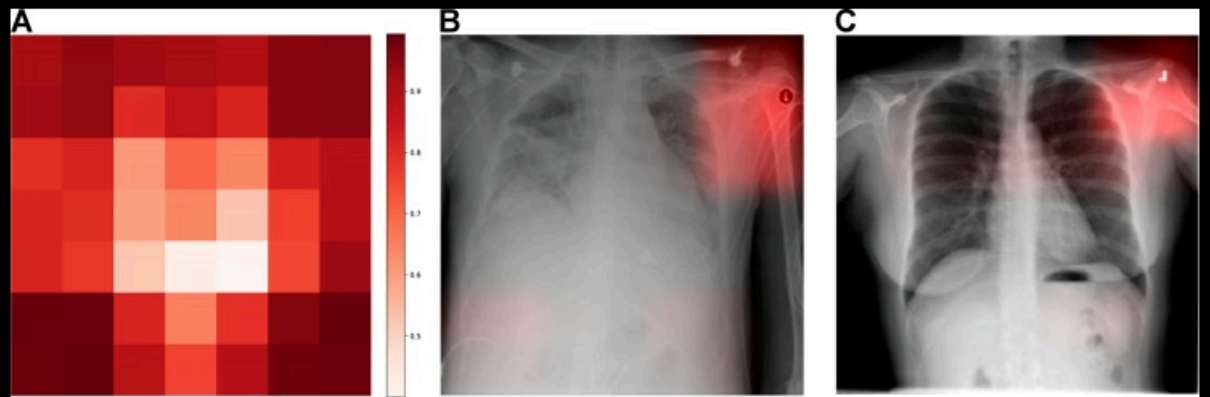
How do you ensure your program « works well »?

What are the most common bugs/malfunctions you encounter?

How do you ensure your program « works well »?

What are the most common bugs/malfunctions you encounter?

- numerical instabilities that invalidated two submissions we tried to reproduce
- debugging shapes : funniest thing to do ever 🤠

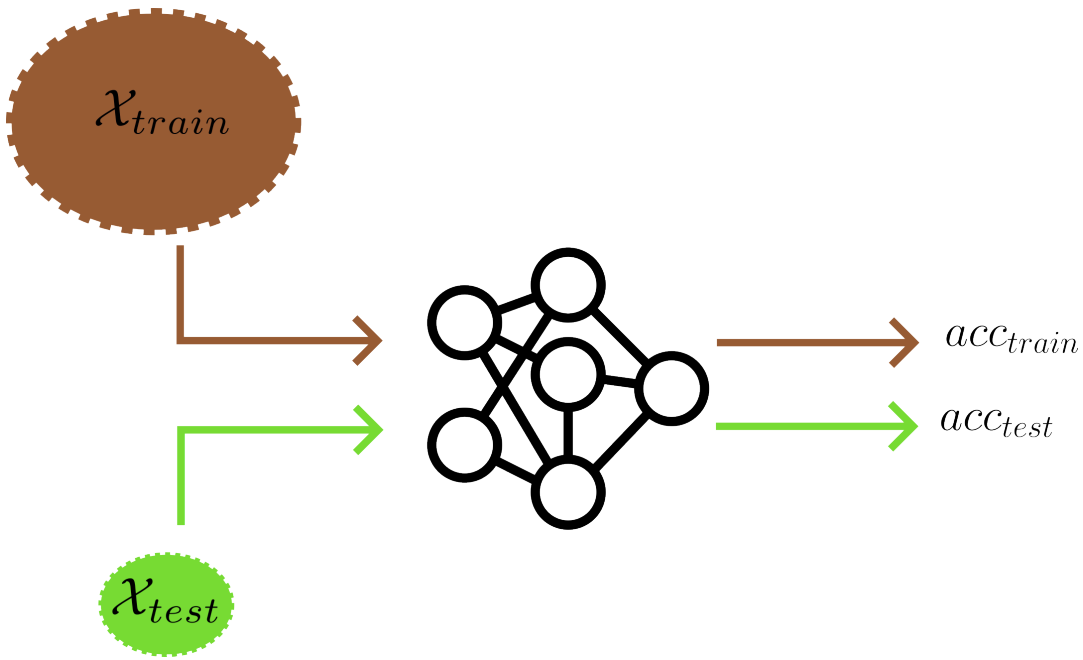


See this image and copyright information in PMC

Fig 2. CNN to predict hospital system detects both general and specific image features. (A) We obtained activation heatmaps from our trained model and averaged over a sample of images to reveal which subregions tended to contribute to a hospital system classification decision. Many different subregions strongly predicted the correct hospital system, with especially strong contributions from image corners. (B-C) On individual images, which have been normalized to highlight only the most influential regions and not all those that contributed to a positive classification, we note that the CNN has learned to detect a metal token that radiology technicians place on the patient in the corner of the image field of view at the time they capture the image. When these strong features are correlated with disease prevalence, models can leverage them to indirectly predict disease. CNN, convolutional neural network.

(Zech et al. 2018) 😊 ← actually dead inside

First introduction of testing on ML



$\delta = \text{acc}_{\text{train}} - \text{acc}_{\text{test}}$ (replace accuracy by other suitable metric in RL or unsupervised learning)

First introduction of testing on ML

Testing a program

The activity of running a program on a predefined scenario, and assess the program's output against an expected behaviour (an *oracle*)

- which scenarios?
  - how do they generalize?
- what is the expected behaviour under test?
- how can the test exhibit and help us understand a program failure?

First introduction of testing on ML

Testing a program

The activity of running a program on a predefined scenario, and assess the program's output against an expected behaviour (an *oracle*)

- which scenarios?
  - **on unseen test data**
  - how do they generalize?
- what is the expected behaviour under test?
- how can the test exhibit and help us understand a program failure?

First introduction of testing on ML

Testing a program

The activity of running a program on a predefined scenario, and assess the program's output against an expected behaviour (an *oracle*)

- which scenarios?
  - **on unseen test data**
  - how do they generalize?
    - **good question!**
- what is the expected behaviour under test?

- how can the test exhibit and help us understand a program failure?

First introduction of testing on ML

Testing a program

The activity of running a program on a predefined scenario, and assess the program's output against an expected behaviour (an *oracle*)

- which scenarios?
  - **on unseen test data**
  - how do they generalize?
    - **good question!**
- what is the expected behaviour under test?
  - **controlled metric degradation**
- how can the test exhibit and help us understand a program failure?

First introduction of testing on ML

Testing a program

The activity of running a program on a predefined scenario, and assess the program's output against an expected behaviour (an *oracle*)

- which scenarios?
  - **on unseen test data**
  - how do they generalize?
    - **good question!**
- what is the expected behaviour under test?
  - **controlled metric degradation**
- how can the test exhibit and help us understand a program failure?
  - **identify spurious counterexample?**

Testing for neural network - the question of scenario

Crucial component of testing is to identify the correct scenarios

Since the search space is enormous, one needs to find relevant scenarios (accidents are hopefully rare in a dataset drawn from real life)

Testing for neural network - the question of scenario

Crucial component of testing is to identify the correct scenarios

Since the search space is enormous, one needs to find relevant scenarios (accidents are hopefully rare in a dataset drawn from real life)

Generating scenarios!

Testing for neural network - the question of scenario

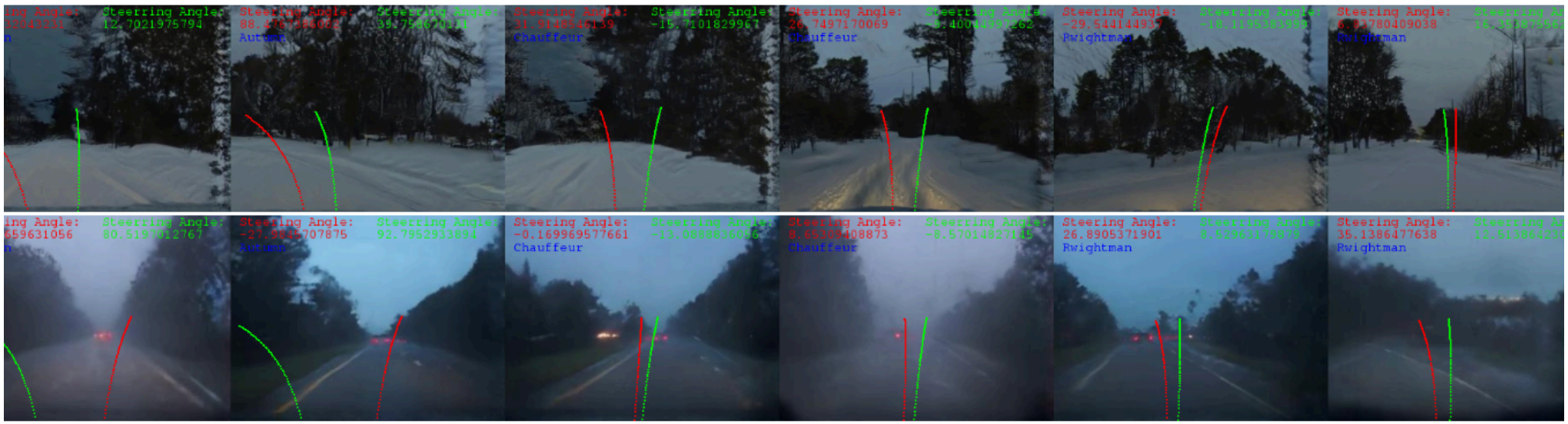


Figure 8: Inconsistency of steering angle prediction on real and synthesized images.

Impact of generated scenarios (from (Zhang et al. 2018))

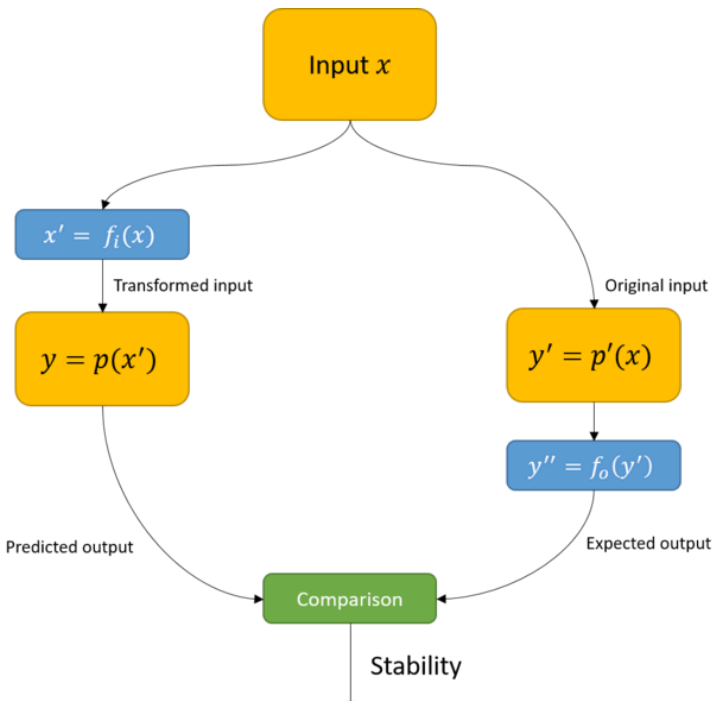
Other examples (Pei et al. 2017; Sun et al. 2018; Tian et al. 2017; Dola, Dwyer, and Soffa 2021; Adjed et al. 2022)

Another approach: Metamorphic testing

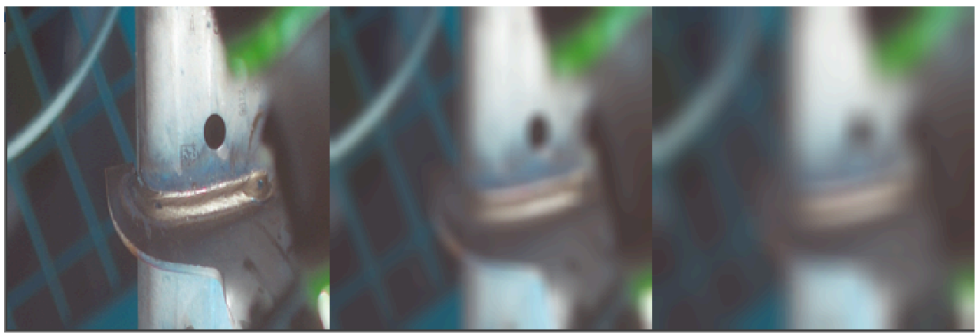
Certain relationships (e.g., R_1, R_2) on some inputs (e.g., a, b, c) should induce, in a software S, other relationships (e.g., R'_1, R'_2)

$$\forall a, b, c, R_1(a, b) \wedge R_2(b, c) \Rightarrow R'_1(S(a), S(b)) \vee R'_2(S(b), S(c)$$

Example: computing the shortest path between two nodes on an undirected graph should be impervious to symmetry



Another approach: Metamorphic testing

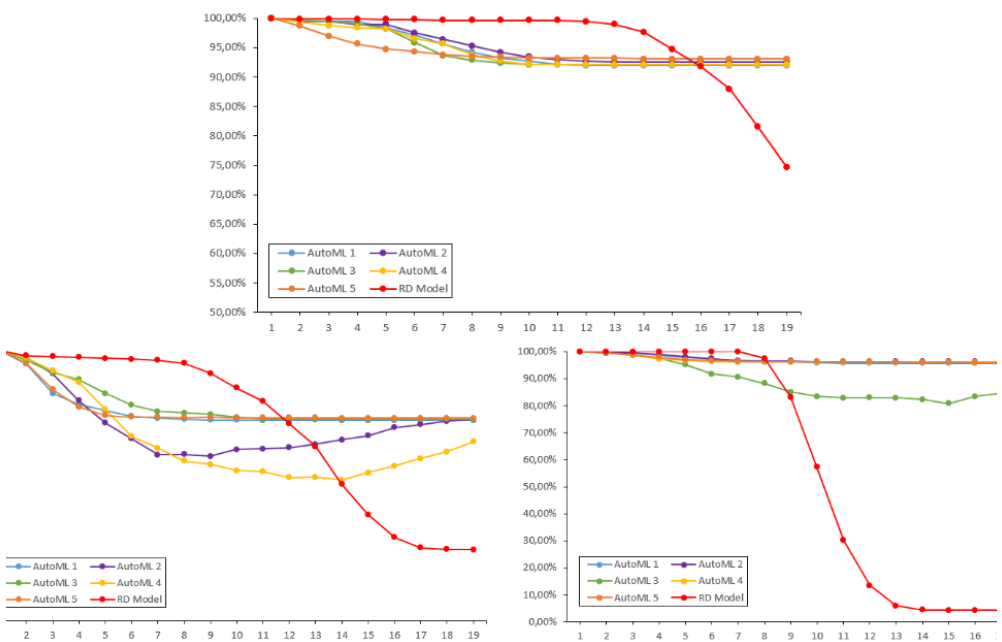


Blur images are not expected to be under the system scope

Metamorphic testing with AIMOS

AIMOS: AI Metamorphic Observing Software (Lemesle et al. 2023)

Assess the stability of a welding production line (in particular: able to spot that a model was performing *too well outside of its dedicated scope*)



Stability of the models on the C10, C20 and C34 welds for the blur proper size ranging from 1 to 20. Value points are linked *only* for aesthetic consid rove readability as AIMOS does not perform interpolation.

Metamorphic testing with AIMOS

Try AIMOS at https://caisar-platform.com/aimos-demo

Metamorphic testing with AIMOS

(a) Original driving scene

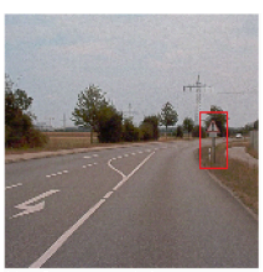
(b) Original driving scene

(c) Original driving scene

(d) Original driving scene

(e) Original driving scene

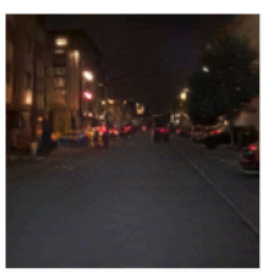
(f) Add a pedestrian



(g) Add a slow sign

(h) Remove a lane line

(i) Replace buildings with trees



(j) Transform to a night scene

A declarative framework for Metamorphic Testing (Deng et al. 2023) with automated rule inference from natural language and generative AI for scenario generation

Industrial tools for testing ML

MLflow: one state-of-the-art platform for evaluating and monitoring machine learning

« Program testing can be used very effectively to show the presence of bugs but never to show their absence.» (Dijkstra 1976)

On the interest of bugfixing with formal verification

Bugfixing?

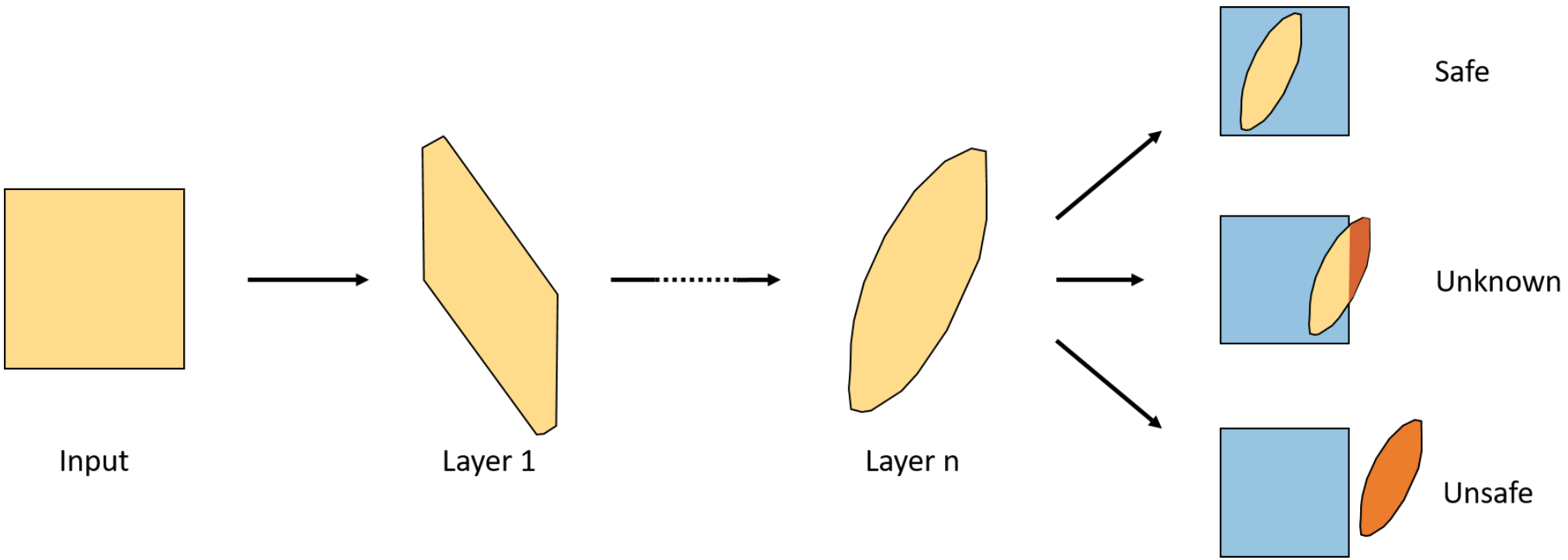
Bugfixing neural networks

Given an input x, a neural network f, a faulty prediction $f(x) = y_{\text{false}}$ and an expected prediction y_{true}, *bugfixing* means constructing a new f' such that $f'^{(x)} = y_{\text{true}}$

Why bugfixing in the first place?

- Retraining may be too costly
- Some behaviours must be preserved
- Particular datapoints must absolutely not fail

Formal verification to the rescue



Definition

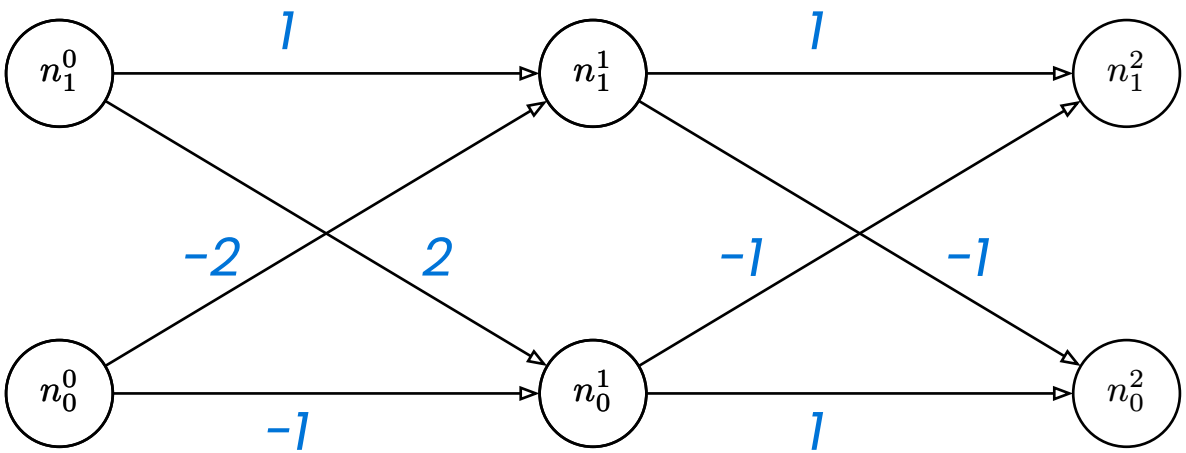
Guaranteed Neural Network repair (Goldberger et al. 2020)

Given a NN $f : x \rightarrow y$, a collection of inputs $\mathcal{X} = \{x_1, ..., x_p\}$, a precondition that holds on all inputs $P(\mathcal{X})$ and a postcondition on all outputs $Q(\mathcal{Y})$, find a new DNN f' such that $P(\mathcal{X}) \models Q(\mathcal{Y})$ and that the *distance* between f and f' is minimal.

Note that, as with local robustness (see Session 2), this property is *local*

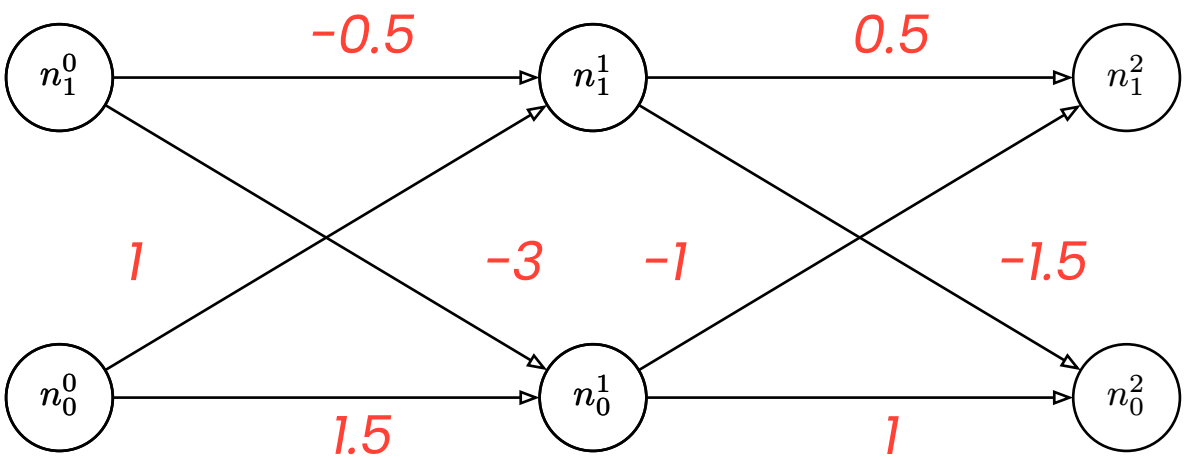
Exemple

Computing the distance between f ...



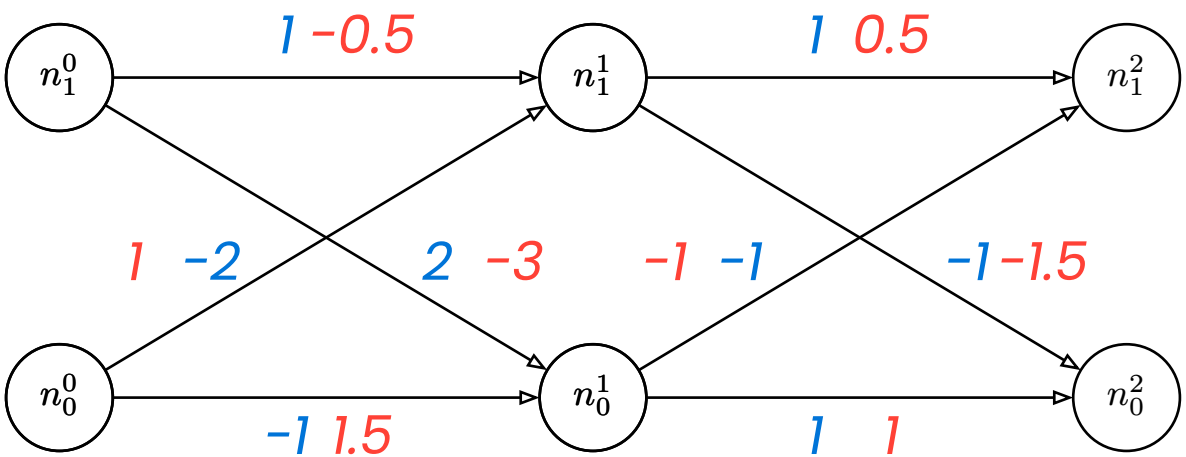
Exemple

... and f':



Exemple

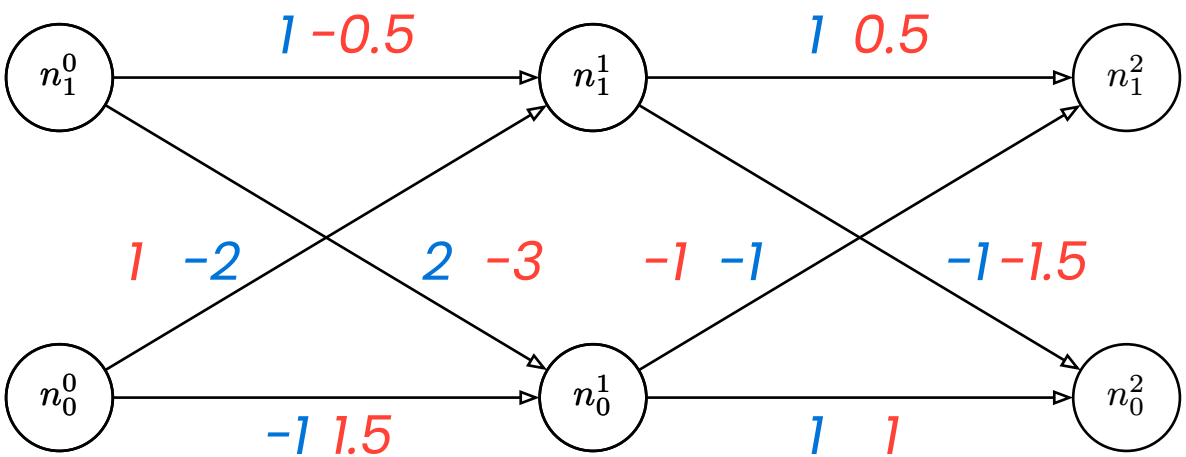
... and f':



$$\|f - f^{\{\prime\}}\|_L = \sum_{\{i=2\}}^{n} \sum_{\{j=1\}} \sum_{\{k=1\}} |W_{i[j,k]} - W_i^{\{\prime\}}[j,k]|^L$$

Exemple

... and f':



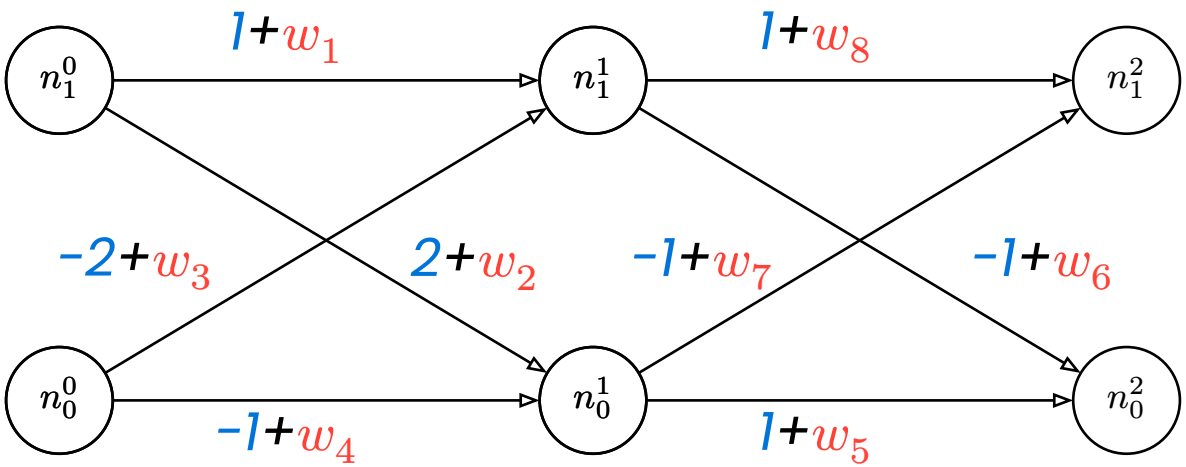
$$\|f - f^{\{\prime\}}\|_L = \sum_{\{i=2\}}^{n} \sum_{\{j=1\}} \sum_{\{k=1\}} |W_{i[j,k]} - W_i^{\{\prime\}}[j,k]|^L$$

$$|1 - 1.5| + |-2 + 3| + |2 - 1| + |-1 + 0.5| + |1 - 0.5| + |-1 + 1.5| + |-1 + 1| + |1 - 1| =$$

$$|0.5| + |1| + |1| + |-0.5| + |0.5| + |0.5| + 0 + 0 = 4$$

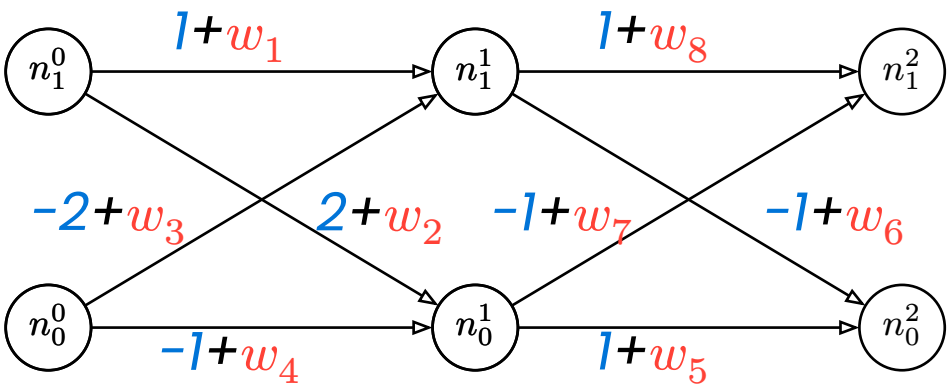
Exemple

Find the best w_i



Search space is now no more the inputs, but the weights

Exemple



$$n_0^1 = \mathrm{ReLU}(1 + w_1)n_1^0 \quad + \mathrm{ReLU}(-2 + w_3)n_0^0$$
$$n_1^1 = \mathrm{ReLU}(2 + w_2)n_1^0 \quad + \mathrm{ReLU}(-1 + w_4)n_0^0$$
$$n_0^2 = \mathrm{ReLU}(1 + w_5)n_0^1 \quad + \mathrm{ReLU}(-1 + w_7)n_1^1$$
$$n_1^2 = \mathrm{ReLU}(-1 + w_6)n_0^1 + \mathrm{ReLU}(1 + w_8)n_1^1$$

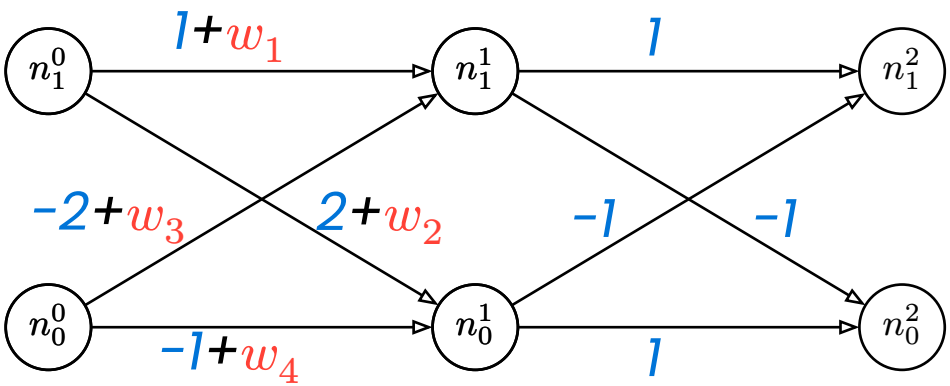
Providing we want to get $n_1^2 \geq n_0^2$, that is to say, ensure that

$$\mathrm{ReLU}(-1 + w_6)n_0^1 + \mathrm{ReLU}(1 + w_8)n_1^1 \geq$$
$$\mathrm{ReLU}(1 + w_5)n_0^1 + \mathrm{ReLU}(-1 + w_7)n_1^1$$
$$=$$
$$\mathrm{ReLU}(-1 + w_6)(\mathrm{ReLU}(1 + w_1)n_1^0 + \mathrm{ReLU}(-2 + w_3)n_0^0) +$$
$$\mathrm{ReLU}(1 + w_8)(\mathrm{ReLU}(2 + w_2)n_1^0 + \mathrm{ReLU}(-1 + w_4)n_0^0) \geq$$
$$\mathrm{ReLU}(1 + w_5)(\mathrm{ReLU}(1 + w_1)n_1^0 + \mathrm{ReLU}(-2 + w_3)n_0^0) +$$
$$\mathrm{ReLU}(-1 + w_7)(\mathrm{ReLU}(2 + w_2)n_1^0 + \mathrm{ReLU}(-1 + w_4)n_0^0)$$

Rephrasing the problem

Key insight: for a given set of inputs, the network is in a *fixed* state

Rephrasing the problem

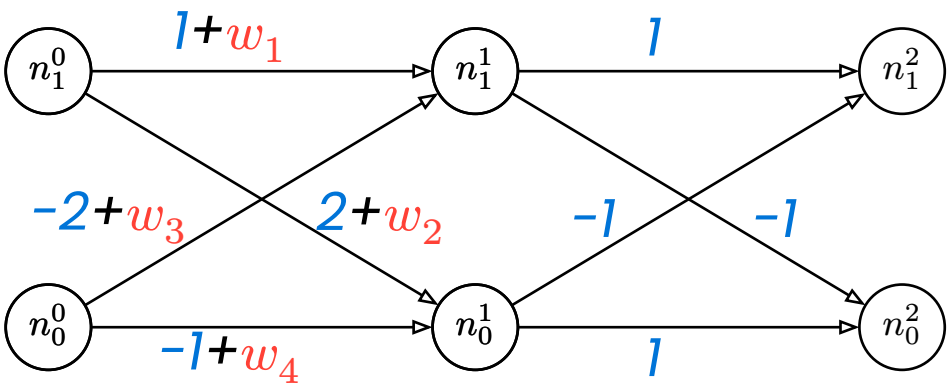


Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$
$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(n_0^1) + \mathrm{ReLU}(-n_1^1)$$
$$n_1^2 = \mathrm{ReLU}(-n_0^1) + \mathrm{ReLU}(n_1^1)$$

Rephrasing the problem

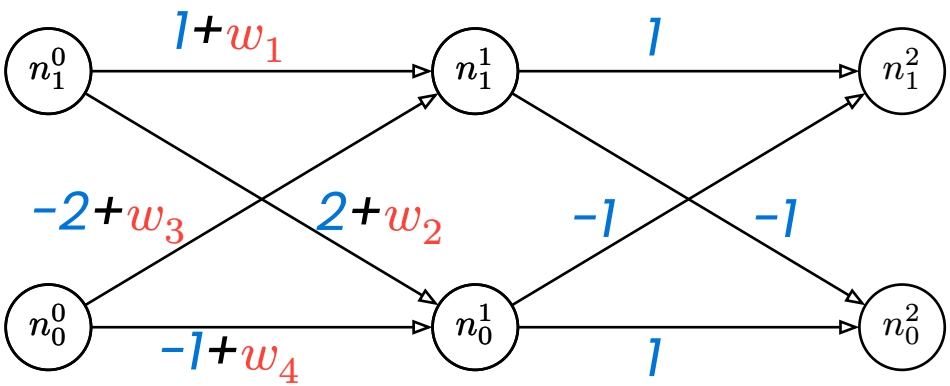


Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$
$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$
$$n_1^2 = -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

Rephrasing the problem



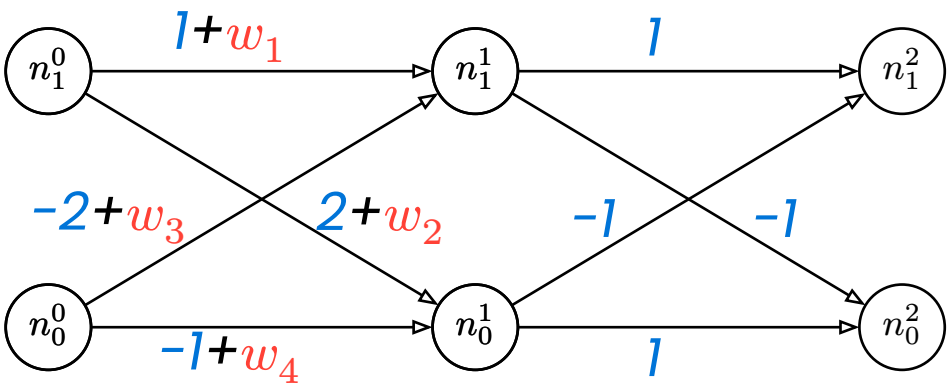
Then, checking whether $n_1^2 \geq n_0^2$ is a problem only in the variables w_i

Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$
$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$
$$n_1^2 = -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

Rephrasing the problem



Then, checking whether $n_1^2 \geq n_0^2$ is a problem only in the variables w_i

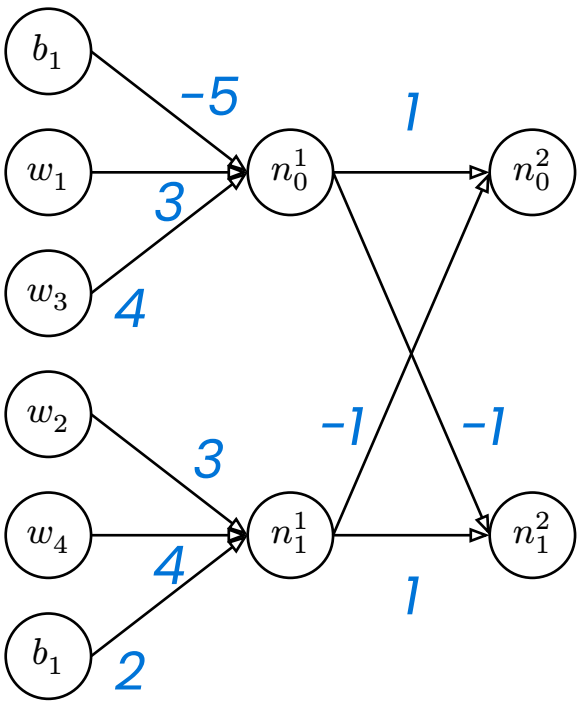
$$P = \bigwedge_{i=1}^{4} -\delta \leq w_i \leq \delta, Q = n_1^2 \geq n_0^2$$

Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$
$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$
$$n_1^2 = -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

Rephrasing the problem



Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$

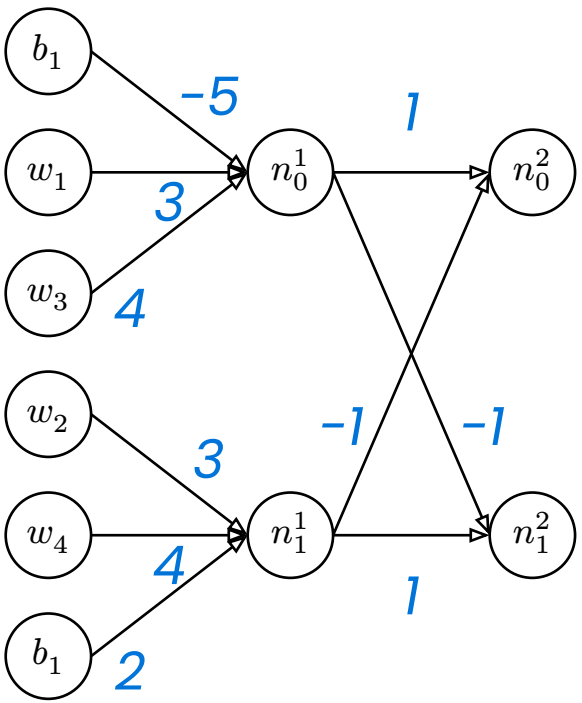
$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

$$n_1^2 = -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

$$P = \bigwedge_{i=1}^{4} -\delta \leq w_i \leq \delta, Q = n_1^2 \geq n_0^2$$

Rephrasing the problem



Given $n_1^0 = 3$ and $n_0^0 = 4$

$$n_0^1 = (1 + w_1)3 + (-2 + w_3)4$$

$$n_1^1 = (2 + w_2)3 + (-1 + w_4)4$$

$$n_0^2 = \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

$$n_1^2 = -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)$$

Local robustness *on the weights!*

$$P = \bigwedge_{i=1}^{4} -\delta \leq w_i \leq \delta, Q = n_1^2 \geq n_0^2$$

Rephrasing the problem

```
Bugfixing NN(network f, Precondition P, Postcondition Q, Collection of inputs X, Layer L):
 1  f' = f
 2  l = []
 3  for x in X:
 4    if CHECK(f',P,Q,x) == SAT then:
 5      APPEND(l,f')
 6    else:
 7      g = BUILD_SYMBOLIC_WEIGHT_NET(f',P,Q,L)
 8      f' = ASSIGN_WEIGHTS(f',w,P,Q)            // NN with weights tricks
 9      CHECK(f',P,Q,x)
10
11  f'' = COMBINE_NETS(l)
12  if CHECK(f'',P,Q):
13    return f''
14  else:
15    RESTART
```

Find the minimal weights perturbations that preserves a neural network behaviour

Initial limitations

Number of water-marks	Average change	Minimal change	Maximal change	Average accuracy	Minimal accuracy	Maximal accuracy
0	0	0	0	0.97	0.97	0.97
1	0.34	0.3	0.38	0.87	0.86	0.88
2	0.43	0.3	1.83	0.79	0.76	0.88
3	0.53	0.33	1.79	0.71	0.66	0.88
4	0.68	0.33	1.79	0.64	0.57	0.88
5	0.79	0.33	1.87	0.59	0.47	0.8
6	0.89	0.35	1.83	0.53	0.38	0.78
7	1.05	0.34	1.91	0.48	0.28	0.78
25	1.86	1.45	2.09	$9.49 \cdot 10^{-2}$	$2.7 \cdot 10^{-3}$	0.41
50	2.05	1.9	2.15	$4.89 \cdot 10^{-2}$	$2.3 \cdot 10^{-3}$	0.2
75	2.13	2.03	2.17	$5.95 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	0.18
100	2.18	2.18	2.18	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$

From (Goldberger et al. 2020). Large modifications result in neural network degradation

Initial limitations

When we ran Algorithm 1 on our ACAS Xu DNN in question, it failed to terminate — we stopped it after several days, while it was attempting to verify φ after the third modification was applied to the network. Thus, while we were unable to verify that all violations of φ was removed from N, we did obtain a network in which at least some of the original violations no longer exist. The results are depicted in Fig. 11. When we examined the counter-examples discovered by the algorithm during its iterations, we observed that they were fairly distant from each other — indicating that the violation is question was not minor, and possibly explaining the difficulty in correcting it.

From (Goldberger et al. 2020). Several days to fix three inputs seems too much…

Initial limitations

Why is so? Because actually encoding « Preserve the accuracy » is a difficult property that cannot be expressed with convex sets!

Initial limitations

We still want to preserve the benefits of formal bugfixing!

- No need to retrain the network
- Guarantee to keep behaviours on given inputs

Main limitations:

- Modifying only one single layer limits the scope of our debugging
- Prover queries can be expensive (See Session 2 on theoretical complexity)

Multi-layer reparation

A neural network f is considered a composition of subnetworks f_k such that $f = f_0 \circ f_1 \circ f_2 ... \circ f_p$. The output layer of f_{k-1} is the input layer of f_k. This layer is present in the original network as L_i.

Multi-layer reparation

A neural network f is considered a composition of subnetworks f_k such that $f = f_0 \circ f_1 \circ f_2 ... \circ f_p$. The output layer of f_{k-1} is the input layer of f_k. This layer is present in the original network as L_i.

Each single layer modification procedure is then launched on the f_ks, their distance against the original f_k is computed and summed up.

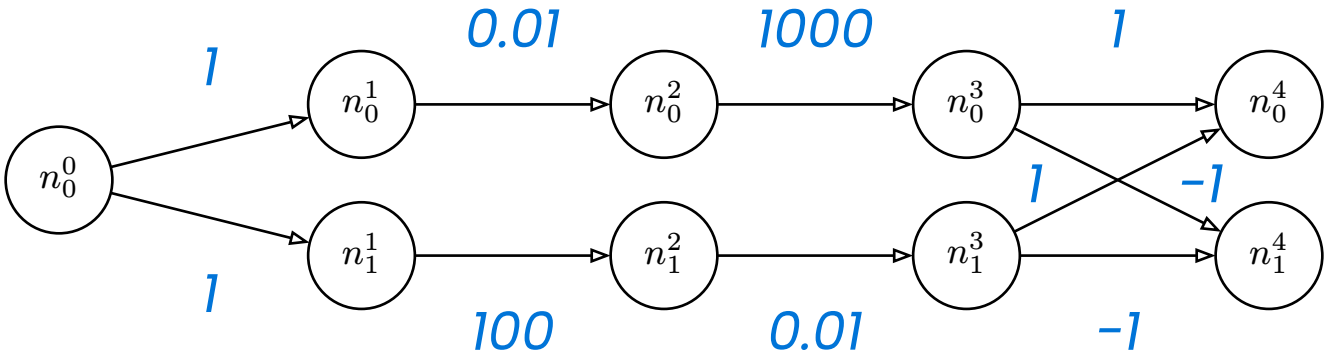
Multi-layer reparation

A neural network f is considered a composition of subnetworks f_k such that $f = f_0 \circ f_1 \circ f_2 ... \circ f_p$. The output layer of f_{k-1} is the input layer of f_k. This layer is present in the original network as L_i.

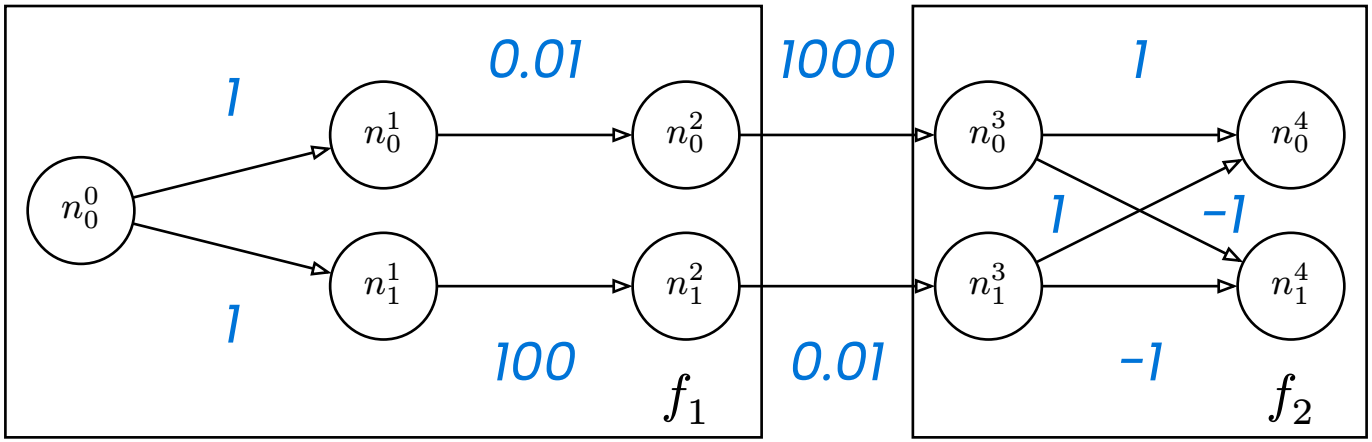
Each single layer modification procedure is then launched on the f_ks, their distance against the original f_k is computed and summed up.

To reuse Single Layer Repair, we need to provide input/output constraints to intermediate layers

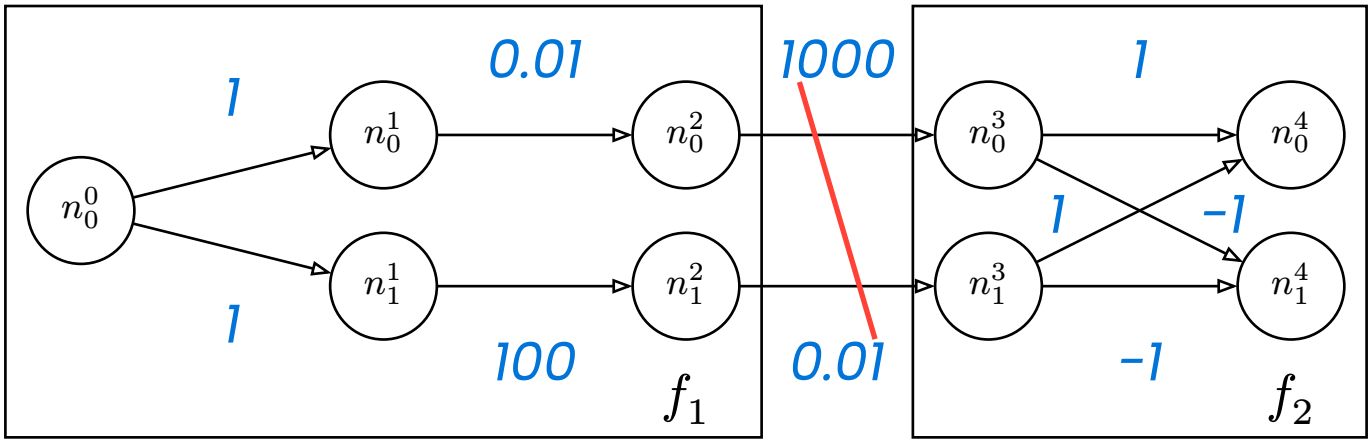
Multi-layer reparation



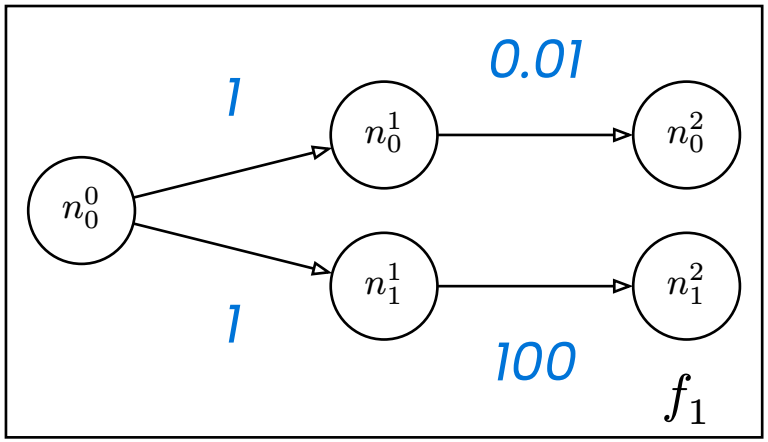
Multi-layer reparation

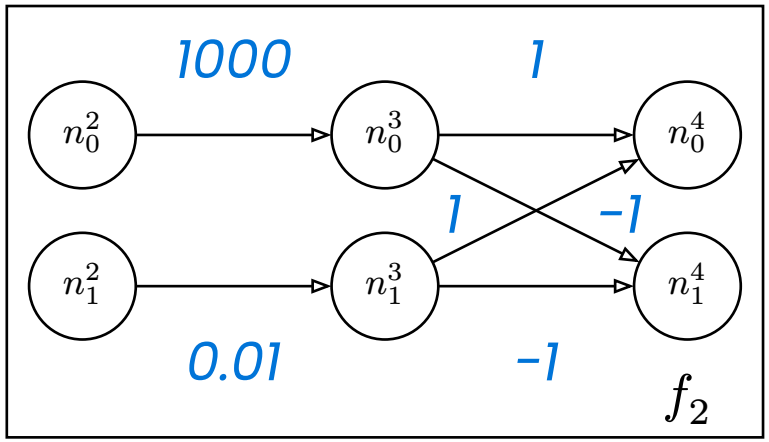


Multi-layer reparation

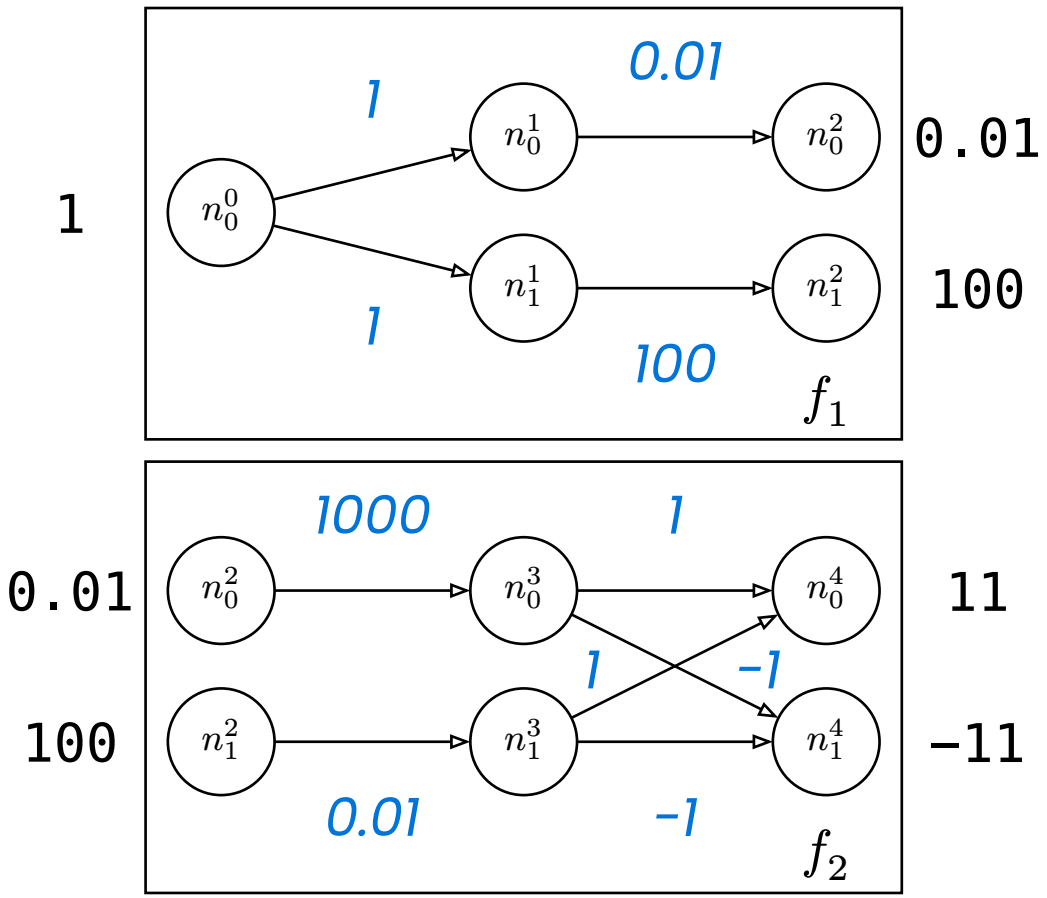


Multi-layer reparation





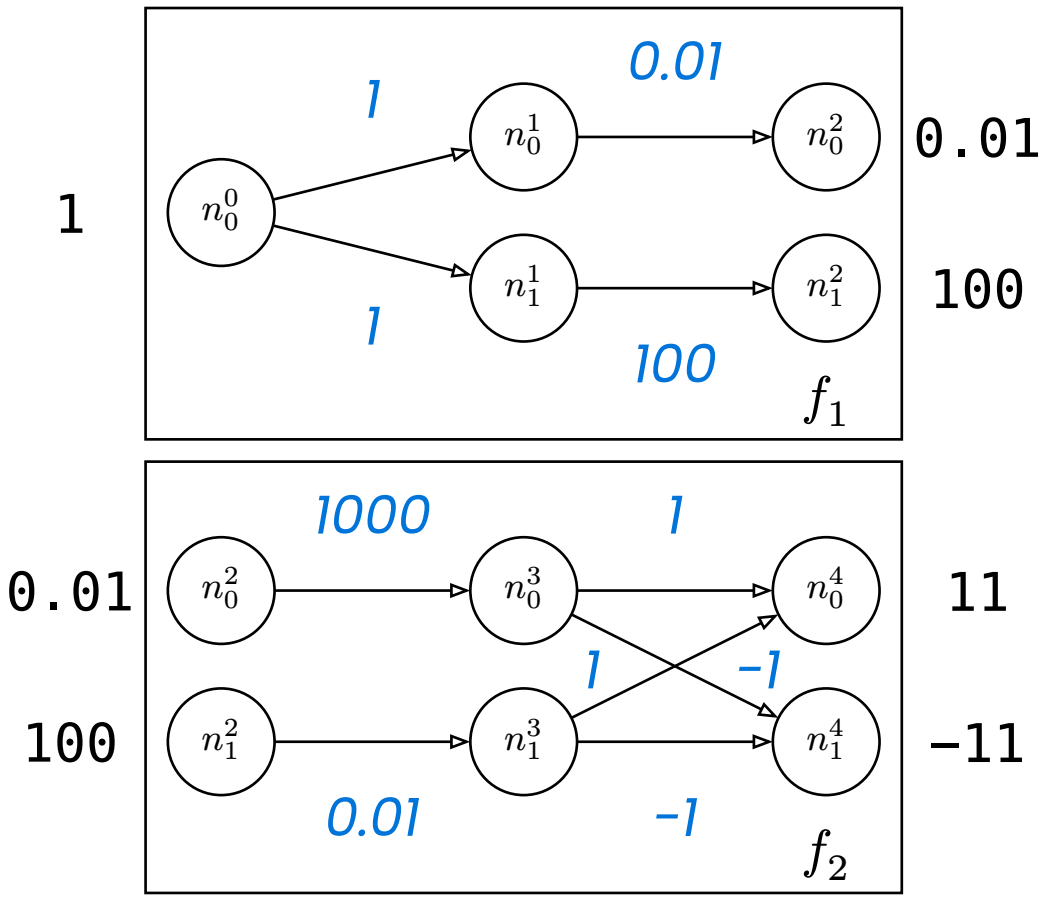
Multi-layer reparation



$n_0^0 = 1 \Rightarrow n_1^4 = 11 > n_0^4 = -11.$

Target behaviour for $n_0^0 : n_1^4 < n_0^4$

Multi-layer reparation



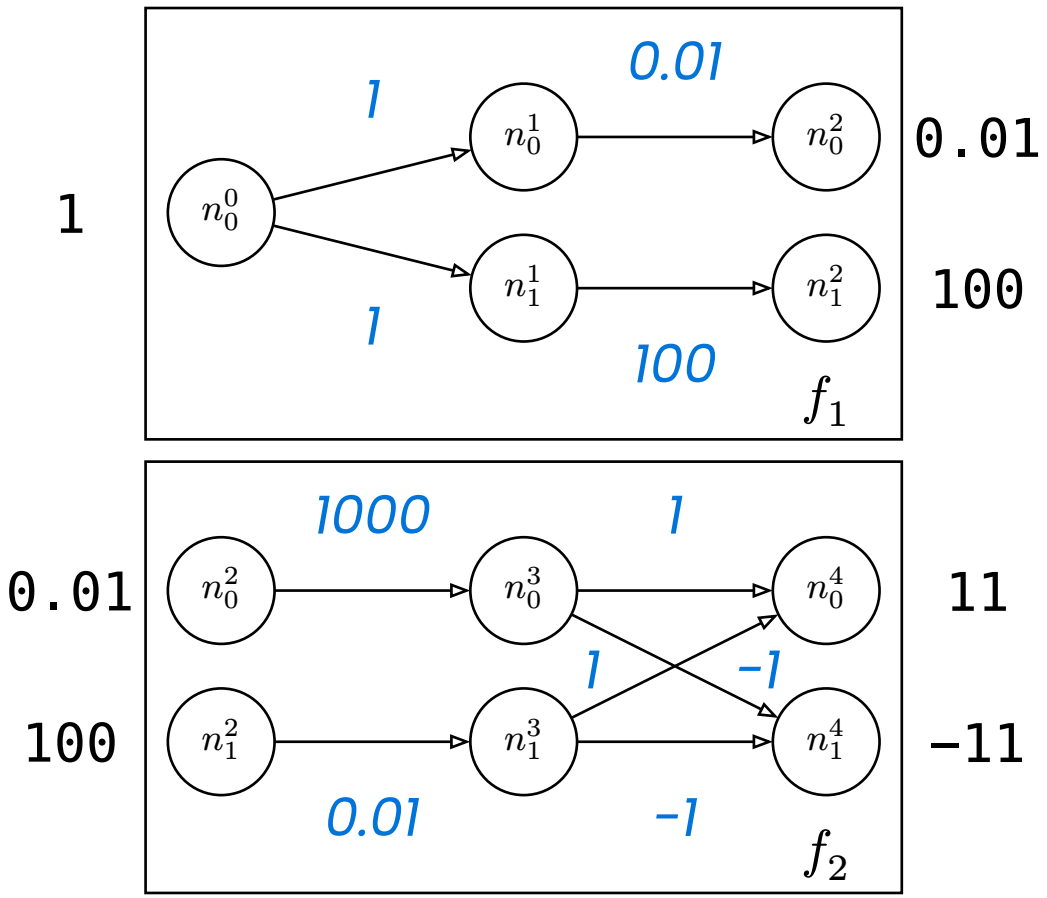
$n_0^0 = 1 \Rightarrow n_1^4 = 11 > n_0^4 = -11.$

Target behaviour for $n_0^0 : n_1^4 < n_0^4$

Three main steps:

1. Assign modified input and outputs bounds to each subnetwork
2. Perform 1-layer verification query to reach the expected behaviour
3. Combine all 1-layer modifications

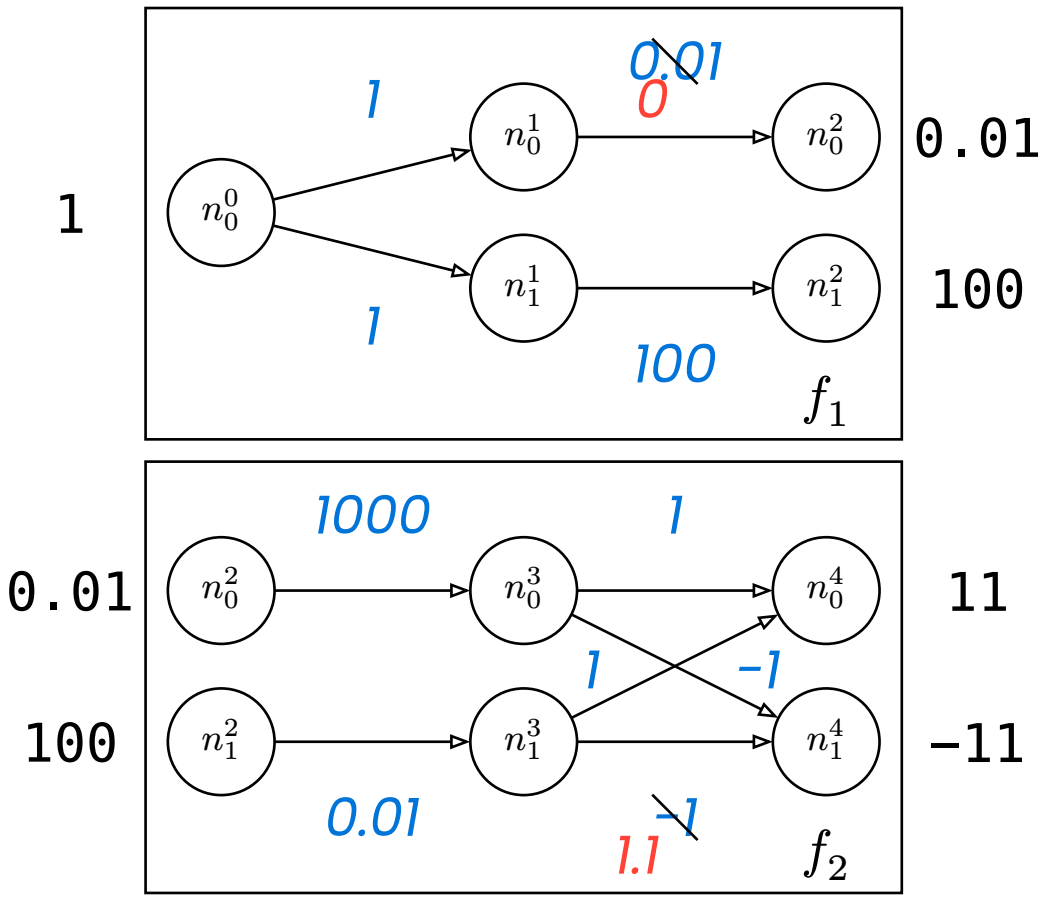
Multi-layer reparation



Target behaviour: $n_1^4 < n_0^4$

1. Propose assignemts:
   - $f_1 : n_0^2 = 0$ (n_1^2 stay unchanged)
   - $f_2 : n_0^4 < n_1^4$
2. Apply Single Layer Repair on f_1 and f_2
3. Merge f_1' and f_2'

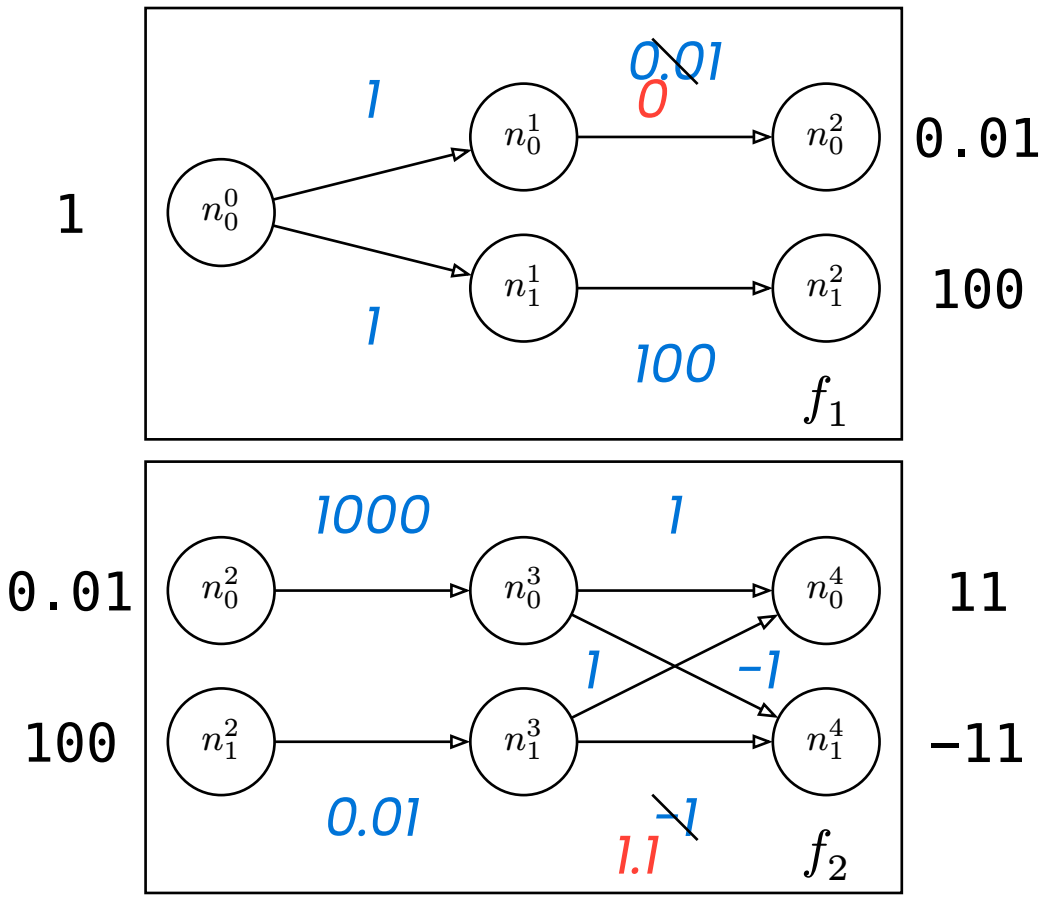
Multi-layer reparation



Target behaviour: $n_1^4 < n_0^4$

1. Propose assignemts:
   - $f_1 : n_0^2 = 0$ (n_1^2 stay unchanged)
   - $f_2 : n_0^4 < n_1^4$
2. Apply Single Layer Repair on f_1 and f_2
3. Merge f_1' and f_2'

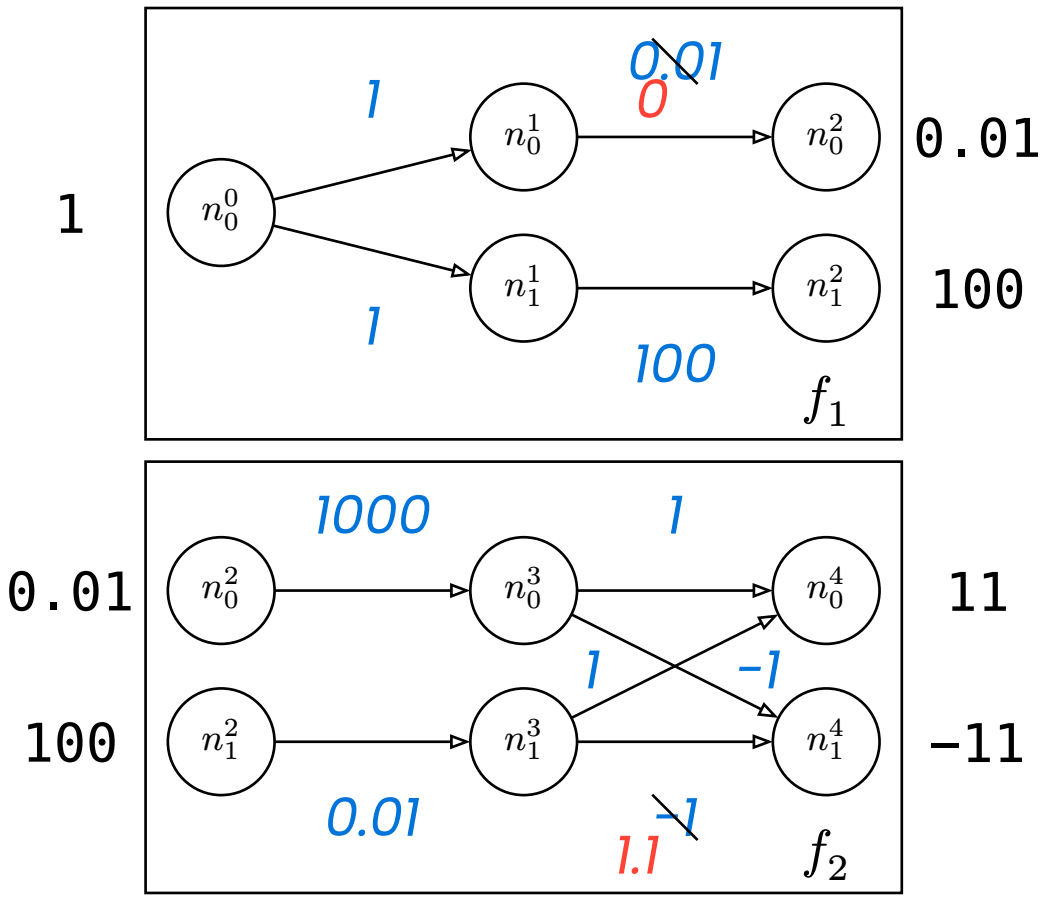
Multi-layer reparation



Target behaviour: $n_1^4 < n_0^4$

1. Propose assignemts:
   - $f_1 : n_0^2 = 0$ (n_1^2 stay unchanged)
   - $f_2 : n_0^4 < n_1^4$
2. Apply Single Layer Repair on f_1 and f_2
3. Merge f_1' and f_2'

Multi-layer reparation



Target behaviour: $n_1^4 < n_0^4$

1. Propose assignemts:
   - $f_1 : n_0^2 = 0$ (n_1^2 stay unchanged)
   - $f_2 : n_0^4 < n_1^4$
2. Apply Single Layer Repair on f_1 and f_2
3. Merge f_1' and f_2'

Multi-layer reparation

```
Multi-layer DNN bugfixing(network f, Collection of inputs 𝒳, Layer indices ℐ, timeout t):
 1  for j in card(𝒳):
 2      assigns = f(x_j)                                  // Compute value assignments for each layer
 3  f_0, ..., f_k = SPLIT(f, ℐ)
 4  best_change, best_cost = ⊥, ∞
 5  while t not exceeded:
 6      for l ∈ ℐ:
 7          c_l = PROPOSE_CHANGE()
 8          for j in card(𝒳):
 9              assigns = assigns + c_l                   // New assignments to prepare SLD
10          for l ∈ ℐ:
11              f'_l, cost_l=SLD(f'_0, < x_0, assigns_0 >, ..., < x_L, assigns_l >)  // Perform Single Layer Debugging
12          cost = TOTAL_COST(cost_0, ..., cost_k)
13          if cost < best_cost:
14              best_cost = cost
15              best_change =< f'_0, ..., f'_k >
16  return best_cost, COMBINE(best_change)
```

Multi-layer reparation

How to propose assignments?

Existing approaches use random search (but could be enhanced)!

Multi-layer reparation

Number of water-marks	Average change	Minimal change	Maximal change	Average accuracy	Minimal accuracy	Maximal accuracy
0	0	0	0	0.97	0.97	0.97
1	0.34	0.3	0.38	0.87	0.86	0.88
2	0.43	0.3	1.83	0.79	0.76	0.88
3	0.53	0.33	1.79	0.71	0.66	0.88
4	0.68	0.33	1.79	0.64	0.57	0.88
5	0.79	0.33	1.87	0.59	0.47	0.8
6	0.89	0.35	1.83	0.53	0.38	0.78
7	1.05	0.34	1.91	0.48	0.28	0.78
25	1.86	1.45	2.09	$9.49 \cdot 10^{-2}$	$2.7 \cdot 10^{-3}$	0.41
50	2.05	1.9	2.15	$4.89 \cdot 10^{-2}$	$2.3 \cdot 10^{-3}$	0.2
75	2.13	2.03	2.17	$5.95 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	0.18
100	2.18	2.18	2.18	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$

From (Goldberger et al. 2020). Large modifications result in neural network degradation

Exp. #	Search Strategy	Number of input points	Average Change	Minimal Change	Maximal Change	Average Accuracy	Minimal Accuracy	Maximal Accuracy
1	Random	1	0.1520	0.0615	0.4922	0.6865	0.1916	0.9308
	Greedy		0.0133	0.001	0.0566	0.943	0.7971	0.9576
	MCTS		0.0139	0.001	0.0566	0.943	0.7971	0.9576
	Random	2	0.197	0.0791	0.4775	0.6302	0.2563	0.9161
	Greedy		0.0463	0.0058	0.1435	0.9245	0.7417	0.9598
	MCTS		0.0478	0.0058	0.1484	0.9261	0.7398	0.9594
2	Greedy 1-Layer	1	0.0305	0.0029	0.1699	0.9397	0.9565	0.5856
			0.0307	0.0029	0.1875	0.9394	0.585	0.9562
	Greedy 1-Layer	2	0.0459	0.0039	0.2041	0.9178	0.3124	0.9576
			0.0464	0.0039	0.208	0.9163	0.3124	0.9576
3	Greedy-3	1	0.25097	0.25097	0.25097	0.886	0.886	0.886

From (Refaeli and Katz 2021) . Proper search space pruning with MCTS result in less performance degradation.

Delta-debugging

Delta-debugging (Zeller and Hildebrandt 2002)

Given an input x and a program f that fails, *delta-debugging* consists on finding the smallest x' such that $f(x')$ still fails.

Useful to reduce the complexity of the input

Somehow related to abductive explanations you saw yesterday in Session 3

Delta-debugging

Application: finding bugs in neural networks verifiers (Elsaleh and Katz 2023)
The input is then the *network* that goes through numerous simplifications/ modifications passes

Obtained neural networks are very *thin*, which shows that neural network verifiers are still yet to mature

Encoding constraints during training

Differentiable Logics

(Ślusarz et al. 2023) being a good introductory paper

Let $\Phi = P(\mathcal{X}) \models Q(\mathcal{Y})$

Given a condition Φ to hold, one wants to generate $\mathcal{L}_\Phi$ such that a network trained with $\mathcal{L} = \mathcal{L}_{\text{acc}} + \mathcal{L}_\Phi$ will both perform its intended purpose *and* respects Φ!

Differentiable Logics

Two components:

- encode Φ (`QF_LRA`? Convex sets? More complex logical languages[1]?)
- interpret Φ as an actual loss $\mathcal{L}_\Phi$

[1] See Benedikt and Daniel's course, or the Datalog one

Differentiable Logics

What we would need: write local robustness under Φ

- defines vectors
- unbounded quantifiers
- defined domains for propositional variables

Differentiable Logics

(Ślusarz et al. 2023) provides a *Logics for Differentiable Logics* that provides automated translation of logical formulaes to actual function losses

Implemented in the Vehicle (Daggitt et al. 2024) language and tool

Differentiable Logics

$$\mathcal{J}(a_1 < a_2) := 1 - \max\left(\frac{a_1 - a_2}{a_1 + a_2}, 0\right)$$

$$\mathcal{J}(p_1 \wedge \mathrm{p}_2) := \mathcal{J}(p_1) * \mathcal{J}(p_2)$$

$$\mathcal{J}(p_1 \Rightarrow p_2) := 1 - \mathcal{J}(p_1) + \mathcal{J}(p_1) * \mathcal{J}(p_2)$$

Thus, local robustness would be:

$$\mathcal{J}(|f(x) - f(x')| \leq \delta) = 1 - \max\left(\frac{|f(x) - f(x')| - \delta}{(|f(x) - f(x')| + \delta)}, 0\right)$$

With Vehicle:

```
@property
robust : Vector Bool n
robust = foreach i . robustAround
  (trainingData ! i) (trainingLabels ! i)
```

Differentiable Logics

$$\mathcal{J}(a_1 < a_2) := 1 - \max\left(\frac{a_1 - a_2}{a_1 + a_2}, 0\right)$$

$$\mathcal{J}(p_1 \wedge p_2) := \mathcal{J}(p_1) * \mathcal{J}(p_2)$$

$$\mathcal{J}(p_1 \Rightarrow p_2) := 1 - \mathcal{J}(p_1) + \mathcal{J}(p_1) * \mathcal{J}(p_2)$$

Thus, local robustness would be:

$$\mathcal{J}(|f(x) - f(x')| \leq \delta) = 1 - \max\left(\frac{|f(x) - f(x')| - \delta}{(|f(x) - f(x')| + \delta)}, 0\right)$$

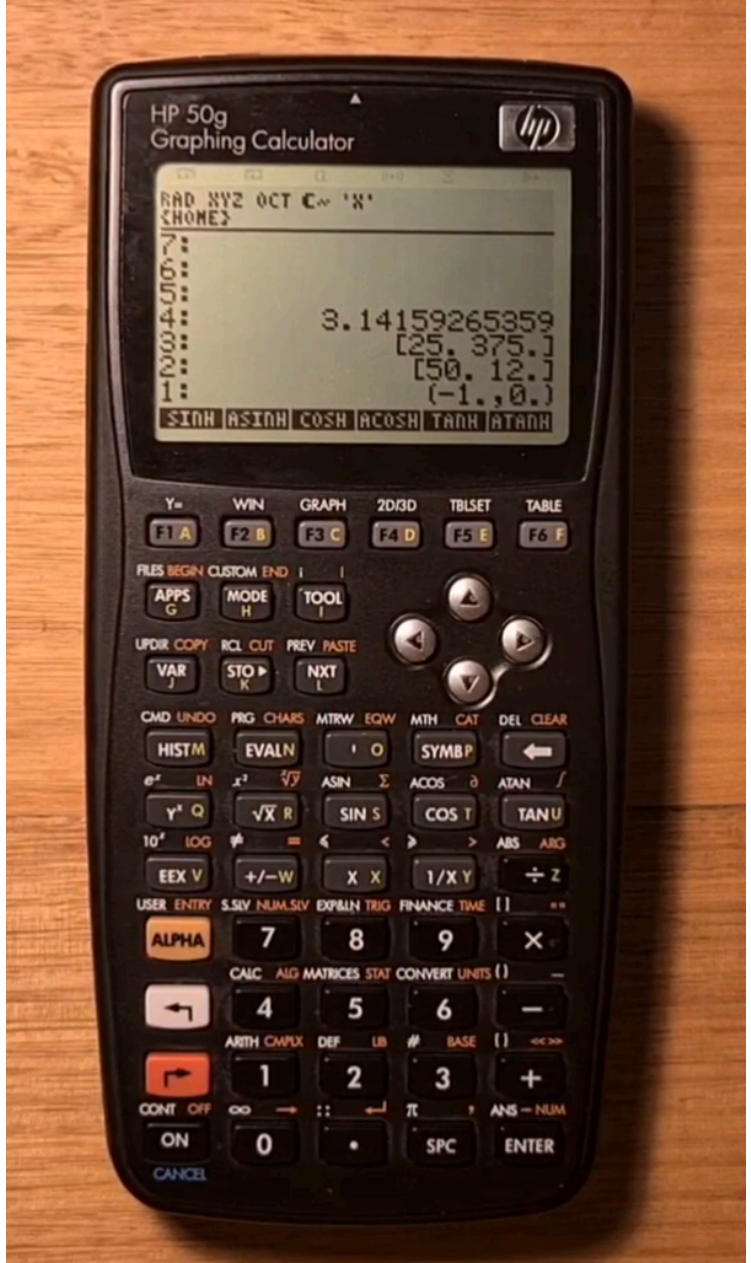
With Vehicle:

```
@property
robust : Vector Bool n
robust = foreach i . robustAround
  (trainingData ! i) (trainingLabels ! i)
```

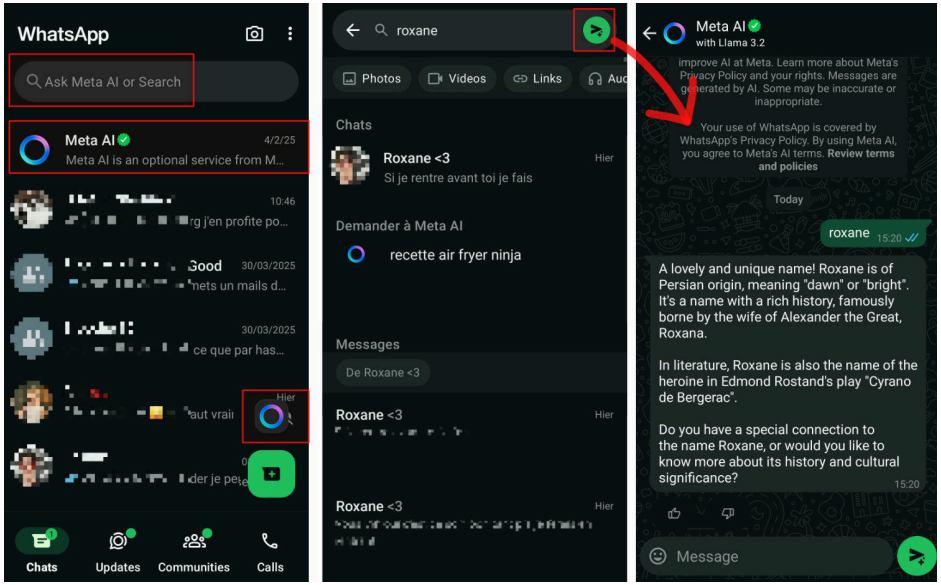
This then can be compiled directly to Python

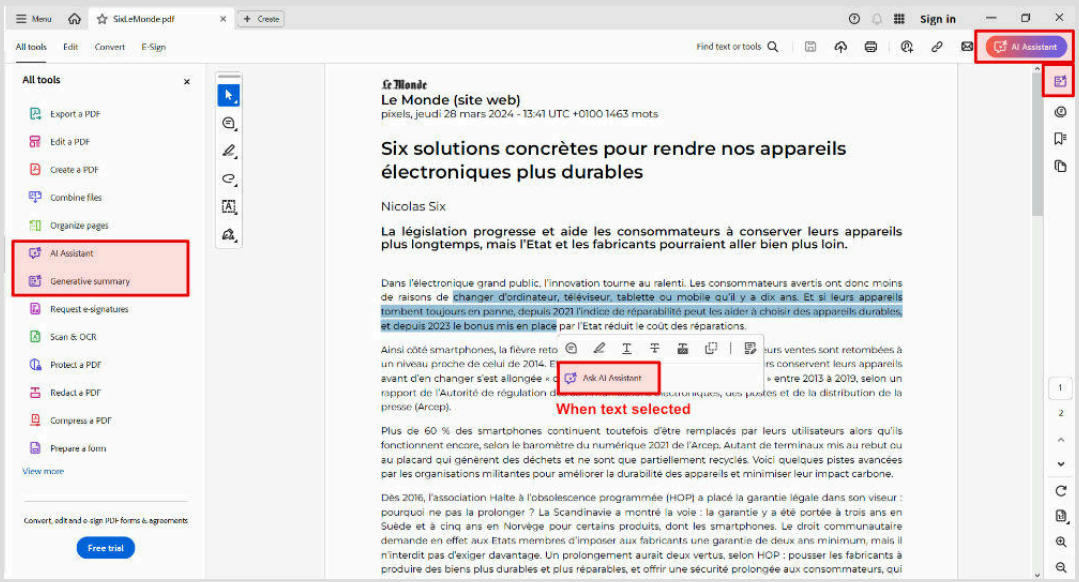
```
from vehicle_lang.compile import Target,
to_python

spec = to_python(
    "mnist-robustness.vcl",
    target=Target.LOSS_DL2,
    samplers={"pertubation":
sampler_for_pertubation},
)

robust_loss_fn = spec["robust"]
```

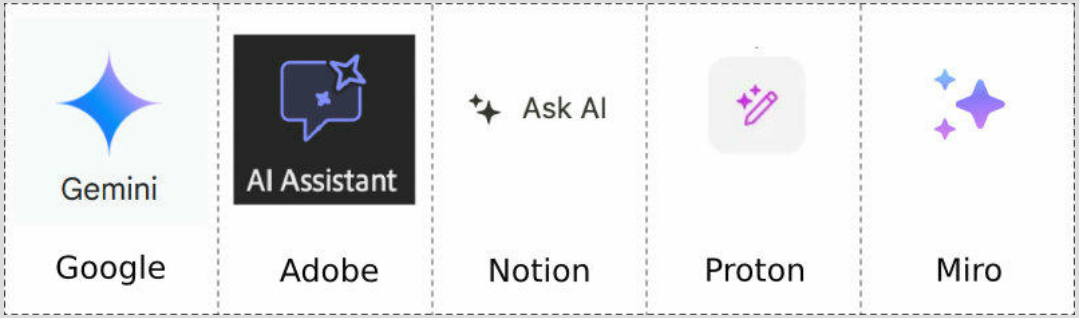
On LLMs and scoping your systems

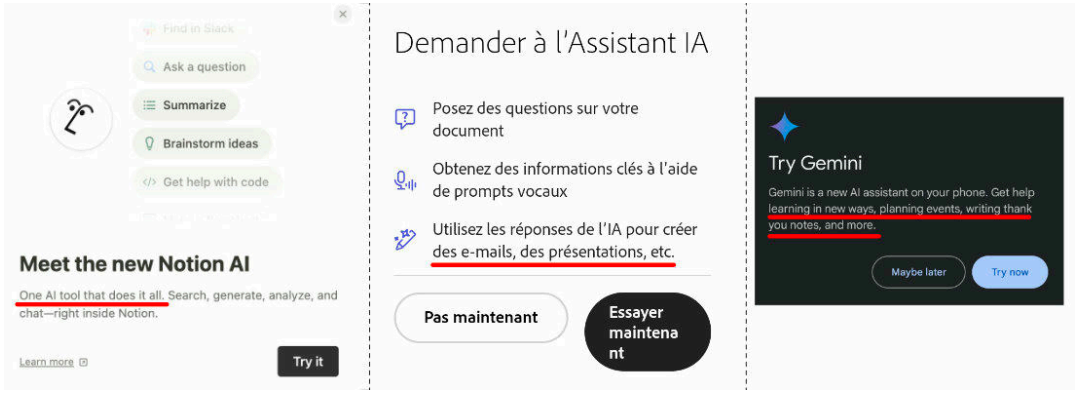

HP 50g
Graphing Calculator
RAD XYZ OCT C↵ 'X'
{HOME}
7:
6:
5:
4: 3.14159265359
3: [25. 375.]
2: [50. 12.]
1: (-1.,0.)
SINH ASINH COSH ACOSH TANH ATANH
Y= WIN GRAPH 2D/3D TBLSET TABLE
F1 A F2 B F3 C F4 D F5 E F6 F
FILES BEGIN CUSTOM END
APPS MODE TOOL
UPDIR COPY RCL CUT PREV PASTE
VAR STO NXT
CMD UNDO PRG CHARS MTRW EQW MTH CAT DEL CLEAR
HISTM EVALN SYMBP
ASIN ACOS ATAN
SIN S COS T TAN U
LOG ABS ARG
EEX V +/-W X X 1/X Y ÷ Z
USER ENTRY S.SLV NUM.SLV EXP&LN TRIG FINANCE TIME
ALPHA 7 8 9 ×
CALC ALG MATRICES STAT CONVERT UNITS
4 5 6 −
ARITH CMPLX DEF LIB # BASE
1 2 3 +
CONT OFF ANS NUM
ON 0 • SPC ENTER
CANCEL

Source of all following image: *https://limitesnumeriques.fr/travaux-productions/ai-forcing* (in french)

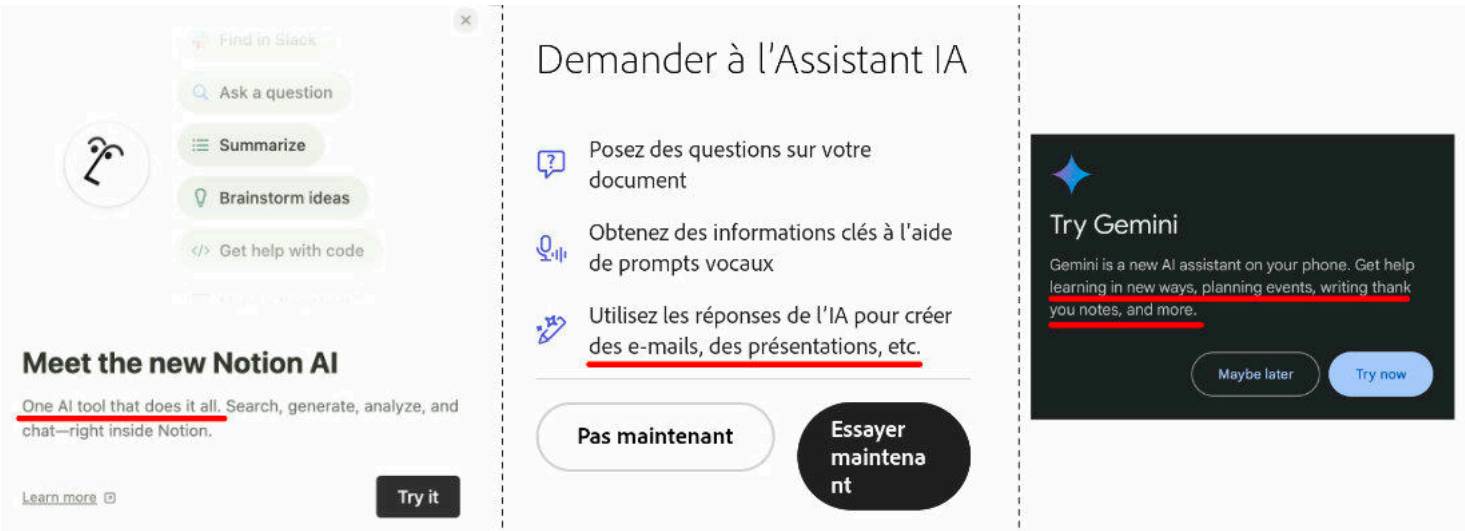

WhatsApp
Ask Meta AI or Search
Meta AI
Meta AI is an optional service from M...
4/2/25
10:46
Good
30/03/2025
30/03/2025
Hier
Chats
Updates
Communities
Calls
roxane
Photos
Videos
Links
Chats
Roxane <3
Hier
Si je rentre avant toi je fais
Demander à Meta AI
recette air fryer ninja
Messages
De Roxane <3
Roxane <3
Hier
Roxane <3
Hier
Meta AI
with Llama 3.2
improve AI at Meta. Learn more about Meta's Privacy Policy and your rights. Messages are generated by AI. Some may be inaccurate or inappropriate.
Your use of WhatsApp is covered by WhatsApp's Privacy Policy. By using Meta AI, you agree to Meta's AI terms. Review terms and policies
Today
roxane
15:20
A lovely and unique name! Roxane is of Persian origin, meaning "dawn" or "bright". It's a name with a rich history, famously borne by the wife of Alexander the Great, Roxana.
In literature, Roxane is also the name of the heroine in Edmond Rostand's play "Cyrano de Bergerac".
Do you have a special connection to the name Roxane, or would you like to know more about its history and cultural significance?
15:20
Message


Sign in
AI Assistant
All tools
Export a PDF
Edit a PDF
Create a PDF
Combine files
Organize pages
AI Assistant
Generative summary
Request e-signatures
Scan & OCR
Protect a PDF
Redact a PDF
Compress a PDF
Prepare a form
View more
Convert, edit and e-sign PDF forms & agreements
Free trial
Le Monde (site web)
pixels, jeudi 28 mars 2024 - 13:41 UTC +0100 1463 mots
Six solutions concrètes pour rendre nos appareils électroniques plus durables
Nicolas Six
La législation progresse et aide les consommateurs à conserver leurs appareils plus longtemps, mais l'Etat et les fabricants pourraient aller bien plus loin.
Ask AI Assistant
When text selected


Gemini
Google
AI Assistant
Adobe
Ask AI
Notion
Proton
Miro


Ask a question
Summarize
Brainstorm ideas
Get help with code
Meet the new Notion AI
One AI tool that does it all. Search, generate, analyze, and chat—right inside Notion.
Learn more
Try it
Demander à l'Assistant IA
Posez des questions sur votre document
Obtenez des informations clés à l'aide de prompts vocaux
Utilisez les réponses de l'IA pour créer des e-mails, des présentations, etc.
Pas maintenant
Essayer maintenant
Try Gemini
Gemini is a new AI assistant on your phone. Get help learning in new ways, planning events, writing thank you notes, and more.
Maybe later
Try now

Requirements (or the lack of thereof)



Requirements (or the lack of thereof)

See how Copilot can be used to help for software engineering

- https://github.com/dotnet/runtime/pull/115762
- https://github.com/dotnet/runtime/pull/115743
- https://github.com/dotnet/runtime/pull/115733

Requirements (or the lack of thereof)

Conflation of words between « AI Safety » and « Software Risks »

> *The term "safety" has come to have a multitude of definitions within AI, which vary based on the context and the community. These definitions have not fully captured the broader meaning of "safety" used within the fields of Systems and Safety Engineering, and may in fact be a direct contradiction to it. Within the context of AI communities, some have defined "safety" as the prevention of failures due to accidents, while others refer to the field of Alignment, aiming to steer AI systems toward human-oriented values and goals. Not only are Alignment measures subjective at best, but they fundamentally conflate safety properties with system requirements, which are well-established engineering concepts.*
>
> — From (Khlaaf 2023)

Requirements (or the lack of thereof)

- Unscoped programs cannot be properly tested
- Most of LLMs discourse in the industry *does not define bounds on the system operation*
- Some work on Software Engineering: *Operational Design Domain*[1]

[1] See https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022R1426

Requirements (or the lack of thereof)

Why GenAI Breaks Traditional MLOps

Traditional machine learning follows predictable patterns. You have datasets with ground truth labels, metrics that clearly indicate success or failure, and deployment pipelines that scale horizontally. GenAI is disruptive not only for its powerful features, but also for introducing foundational changes to how quality and stability are measured and ensured.

Consider a simple question: "How do you know if your RAG system is working correctly?" In traditional ML, you'd check accuracy against a test set. In GenAI, you're dealing with:

Complex execution flows involving multiple LLM calls, retrievals, and tool interactions

Subjective output quality where "correct" can mean dozens of different valid responses

Latency and cost concerns that can make or break user experience

Debugging nightmares when something goes wrong deep in a multi-step reasoning chain

The current solution? Most teams cobble together monitoring tools, evaluation scripts, and deployment pipelines from different vendors. The result is fragmented workflows where critical information gets lost between systems.

MLflow comment on GenAI

⚠️ 🔥 Personal Take Alarm 🔥 ⚠️

Only my personal opinion on the next slide
Please discuss

- Current LLMs design principles and implementations are **adverse for verification and proper software engineering**
  - Yet they are deployed
- I see little value on systems that are not working, which quality is difficult to assess and control, put a heavy toll on planetary resources and societies
- Hot take steming from (Bender et al. 2021), (Varoquaux, Luccioni, and Whittaker 2025) and (Crawford 2021)

⚠️ 🔥 Personal Take Alarm 🔥 ⚠️

Discussions and future trends

On the importance of a good testing scenario

- Ensuring good coverage on the scenario
- Having a specification of your system operation is crucial (Khlaaf 2023)
  - That is why it is impossible to test LLMs, as they are usually unscoped and unbounded

Formal verification has a card to play

- Checking crucial instances is worthy on critical settings
- Runtime prevents for now of applying this to large networks on a large set of points

Open questions

Testing

- How to specify which scenarios to generate?

Debugging

- Can we « keep repairing » until we achieve 100% accuracy?
- Extend the search to multiple architectures?
  - How to encode that problem?

Logical constraints

- How much of Φ can be compiled to a loss $\mathcal{L}$?
  - Is the network any good at learning the constraint?

Bibliography

Adjed, Faouzi, Mallek Mziou-Sallami, Frédéric Pelliccia, Mehdi Rezzoug, Lucas Schott, Christophe Bohn, and Yesmina Jaafra. 2022. “Coupling Algebraic Topology Theory, Formal Methods and Safety Requirements toward a New Coverage Metric for Artificial Intelligence Models”. *Neural Computing and Applications*, May. Springer Science, Business Media LLC. doi:10.1007/s00521-022-07363-6.

Bender, Emily M., Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜 ”. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, And Transparency*, 610–623. Facct '21. Virtual

Event, Canada: Association for Computing Machinery. doi:10.1145/3442188.3445922.

Crawford, Kata. 2021. *The Atlas of AI: Power, Politics, And the Planetary Costs of Artificial Intelligence*. Yale University Press. doi:10.2307/j.ctv1ghv45t.

Daggitt, Matthew L., Wen Kokke, Robert Atkey, Natalia Slusarz, Luca Arnaboldi, and Ekaterina Komendantskaya. 2024. "Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs". arXiv. doi:10.48550/ARXIV.2401.06379.

Deng, Yao, Xi Zheng, Tianyi Zhang, Huai Liu, Guannan Lou, Miryung Kim, and Tsong Yueh Chen. 2023. "A Declarative Metamorphic Testing Framework

for Autonomous Driving". *IEEE Transactions on Software Engineering* 49 (4): 1964–1982. doi:10.1109/TSE.2022.3206427.

Dijkstra, E.W. 1976. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall.

Dola, Swaroopa, Matthew B. Dwyer, and Mary Lou Soffa. 2021. "Distribution-Aware Testing of Neural Networks Using Generative Models". In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 226–237. doi:10.1109/ICSE43902.2021.00032.

Elsaleh, Raya, and Guy Katz. 2023. "Delbugv: Delta-Debugging Neural Network Verifiers". arXiv. doi:10.48550/ARXIV.2305.18558.

Goldberger, Ben, Guy Katz, Yossi Adi, and Joseph Keshet. 2020. “Minimal Modifications of Deep Neural Networks Using Verification”. In *Lpar23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, edited by Elvira Albert and Laura Kovacs, 73:260–278. Epic Series in Computing. EasyChair. doi:10.29007/699q.

Katz, Guy, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In *Computer Aided Verification*, 97–117. Springer International Publishing. doi:10.1007/978-3-319-63387-9_5.

Khlaaf, Heidy. 2023. *Toward Comprehensive Risk Assessments and Assurance of AI-Based Systems*. https://www.trailofbits.com/documents/Toward_comprehensive_risk_assessments.pdf.

Lemesle, Augustin, Aymeric Varasse, Zakaria Chihani, and Dominique Tachet. 2023. “AIMOS: Metamorphic Testing of AI - AnIndustrial Application”. *WAISE 2023*.

Pei, Kexin, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. “Deepxplore: Automated Whitebox Testing of Deep Learning Systems”. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 1–18. SOSP '17. Shanghai, China: Association for Computing Machinery. doi:10.1145/3132747.3132785.

Refaeli, Idan, and Guy Katz. 2021. “Minimal Multi-Layer Modifications of Deep Neural Networks”. arXiv. doi:10.48550/arXiv.2110.09929.

Sun, Youcheng, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. “Concolic Testing for Deep Neural Networks”. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 109–119. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3238147.3238172.

Tian, Yuchi, Kexin Pei, Suman Jana, and Baishakhi Ray. 2017. “Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars”. *Corr*. http://arxiv.org/abs/1708.08559.

Varoquaux, Gaël, Alexandra Sasha Luccioni, and Meredith Whittaker. 2025. "Hype, Sustainability, And the Price of the Bigger-Is-Better Paradigm in AI". arXiv. doi:10.48550/arXiv.2409.14160.

Zech, John R., Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric Karl Oermann. 2018. "Variable Generalization Performance of a Deep Learning Model to Detect Pneumonia in Chest Radiographs: A Cross-Sectional Study". Edited by Aziz Sheikh. *PLOS Medicine* 15 (11). Public Library of Science (PLoS): e1002683. doi:10.1371/journal.pmed.1002683.

Zeller, A., and R. Hildebrandt. 2002. "Simplifying and Isolating Failure-Inducing Input". *IEEE Transactions on Software Engineering* 28 (2).

Institute of Electrical, Electronics Engineers (IEEE): 183–200. doi:10.1109/32.988498.

Zhang, Mengshi, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. “DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems”. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3238147.3238187.

Ślusarz, Natalia, Ekaterina Komendantskaya, Matthew L. Daggitt, Robert Stewart, and Kathrin Stark. 2023. “Logic of Differentiable Logics: Towards a Uniform Semantics of Dl.”