

Composer une salade CAISAR avec des réseaux de neurones et de l'édition de graphe

Journées Francophone des Langages Applicatifs 2025, Roiffé

Michele Alberti (CEA LIST),

Julien Girard-Satabin (CEA LIST) : julien.girard2@cea.fr

Alban Grastien (CEA LIST), François Bobot (CEA LIST), Aymeric Varasse (now at Software Heritage), Zakaria Chihani (CEA LIST)



- *Une salade composée peut être encore plus savoureuse en mélangeant les légumes crus et cuits. Laissez libre cours à votre imagination : brocolis légèrement grillés, courgettes rôties, asperges sautées... N'ayez pas peur d'expérimenter, c'est la clé pour découvrir de nouvelles **combinaisons délicieuses**.*
- *Ne négligez pas la vinaigrette. La vinaigrette est l'élément clé **qui lie tous les ingrédients de votre salade**.*
- *Une salade sans vinaigrette, c'est comme un labo sans dotation (anonyme de Verimag)*



Source : 750g.com

**Spécification et vérification de
composants d'intelligence
artificielle, c'est pas des salades**

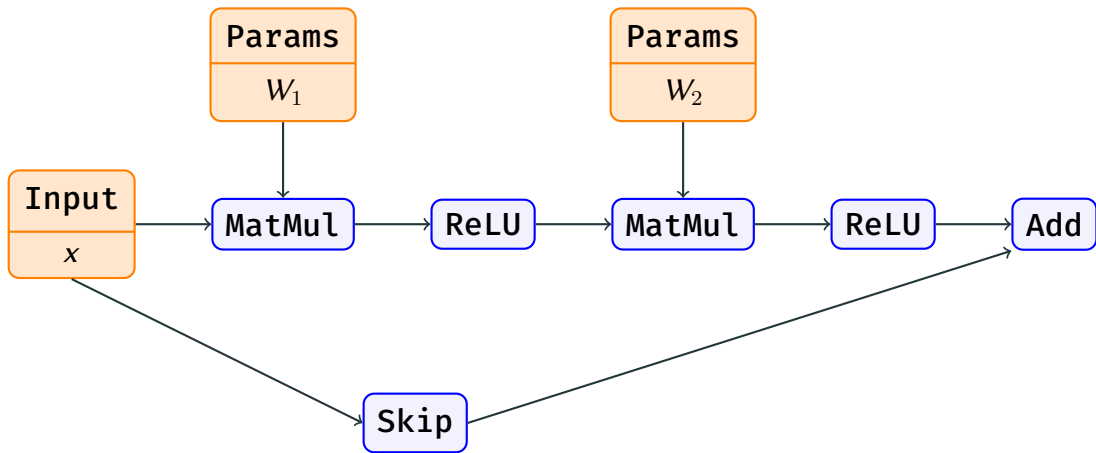
Intelligence artificielle ?

Définition

”Intelligence Artificielle” désigne des programmes obtenus suite à un processus d'apprentissage statistique.

Un programme appris est une succession de calcul arithmétiques paramétrés linéaires et non-linéaires sur des tableaux de flottants.

Machine à vecteur de support, réseau de neurone... *ni boucle ni branchement conditionnel*



Un standard de représentation

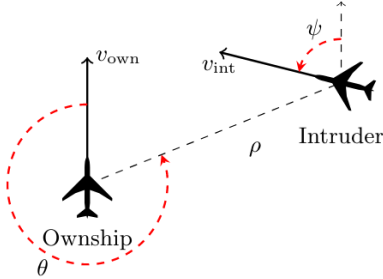
Open Neural Network eXchange format (ONNX) : sémantique de 194 opérations décrite, organisés sous forme de graphe acyclique orienté

ONNX Operators

Lists out all the ONNX operators. For each operator, lists out the usage guide, parameters, examples, and line-by-line version history. This section also includes tables detailing each operator with its versions, as done in [Operators.md](#).

All examples end by calling function `expect`, which checks a runtime produces the expected output for this example. One implementation based on [onnxruntime](#) can be found at [Sample operator test code](#).

ai.onnx	ai.onnx.ml	ai.onnx.preview.training
operator	versions	differences
Abs	13, 6, 1	13/6, 13/1, 6/1
Acos	22, 7	22/7
Acosh	22, 9	22/9
Add	14, 13, 7, 6, 1	14/13, 14/7, 13/7, 14/6, 13/6, 7/6, 14/1, 13/1, 7/1, 6/1
AffineGrid	20	
And	7, 1	7/1
ArgMax	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
ArgMin	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1,



Property ϕ_1 .

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- Tested on: all 45 networks.
- Input constraints: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.
- Desired output property: the score for COC is at most 1500.

Property ϕ_2 .

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- Tested on: $N_{x,y}$ for all $x \geq 2$ and for all y .
- Input constraints: $\rho \geq 55947.691$, $v_{own} \geq 1145$, $v_{int} \leq 60$.
- Desired output property: the score for COC is not the maximal score.

Property ϕ_3 .

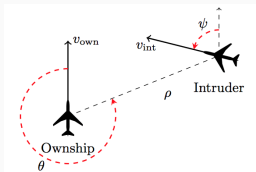
- Description: If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- Tested on: all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.
- Input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{own} \geq 980$, $v_{int} \geq 960$.
- Desired output property: the score for COC is not the minimal score.

Cas d'étude ACAS-Xu. Les valeurs θ , ρ et ψ représentent des angles relatifs entre un appareil *ownship* et un appareil *intrus*; v_{own} la vitesse de l'appareil *ownship* et v_{int} la vitesse de l'intrus. Les propriétés décrivent des comportements attendus de *ownship* par rapport à l'intrus. Adapté de [Kat+17].

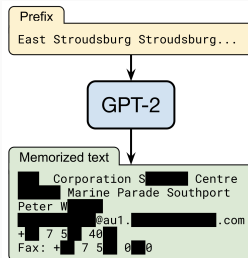
Des programmes notablement difficiles à maîtriser...



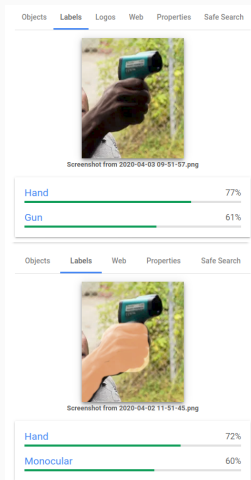
Robustesse locale [Mad+17]?



Propriétés fonctionnelles [Kat+17]?



Respect de vie privée [Car+23])?



Équité [Urb+19]?

... demanderont de gros efforts pour être vérifiés

- PyRAT (c'est nous!)[LLG24]
- Marabou [Kat+19]
- Neurify

... demanderont de gros efforts pour être vérifiés

- PyRAT (c'est nous!)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19; Mül+21]
- $\alpha - \beta$ -Crown [Wan+21]
- NNV^a
- FaceLattice [Yan+21]

... demanderont de gros efforts pour être vérifiés

- PyRAT (c'est nous!)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19; Mül+21]
- $\alpha - \beta$ -Crown [Wan+21]
- NNV^a
- FaceLattice [Yan+21]
- Facet-Vertex incidence^b
- ReluDiff [PWW20]
- Peregrinn [KFS21]

a. <https://github.com/verivital/nnv>

b. <https://github.com/Shaddadi/Facet-Vertex-FFNN>

... demanderont de gros efforts pour être vérifiés

- PyRAT (c'est nous!)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19; Mül+21]
- $\alpha - \beta$ -Crown [Wan+21]
- NNV^a
- FaceLattice [Yan+21]
- Facet-Vertex incidence^b
- ReluDiff [PWW20]
- Peregrinn [KFS21]

- Oval^a
- nnenum [Bak21]
- Libra [Urb+19]

a. <https://github.com/verivital/nnv>

b. <https://github.com/Shaddadi/Facet-Vertex-FFNN>

... demanderont de gros efforts pour être vérifiés

- PyRAT (c'est nous!)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19; Mül+21]
- $\alpha - \beta$ -Crown [Wan+21]
- NNV^a
- FaceLattice [Yan+21]
- Facet-Vertex incidence^b
- ReluDiff [PWW20]
- Peregrinn [KFS21]

- Oval^a
- nnenum [Bak21]
- Libra [Urb+19]
- MIPVerify [TXT19]
- Planet [Ehl17]
- Sherlock [Dut+17]
- ZoPE[Str+22]
- DNNV [SED21]
- Veritex^b
- Verinet and Venus^c

a. <https://github.com/verivital/nnv>

b. <https://github.com/Shaddadi/Facet-Vertex-FFNN>

a. <https://github.com/oval-group/oval-bab>

b. <https://github.com/Shaddadi/veritex>

c. <https://github.com/vas-group-imperial/VeriNet>



(crédit : Bill Wurtz)

Niches écologiques pour prouveurs

Spécialisations

1. par type de spécification : robustesse locale, type de propriété, *fairness*;
2. par techniques employées : Satisfaction Modulo Théorie (SMT), Programmation par contrainte (CP), interprétation abstraite;
3. par modèle ciblé : réseaux de neurones avec ReLU, machines à vecteur de support [RZ19]... et tous les solveurs SMT existant !

Un standard de-facto peu satisfaisant

```

(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)

(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(declare-const Y_3 Real)
(declare-const Y_4 Real)

; Unscaled Input 0 : (36000, 60760)
(assert (<= X_0 0.679857769))
(assert (>= X_0 0.268978427))

; Unscaled Input 1 : (0.7, 3.141592)
(assert (<= X_1 0.499999896))
(assert (>= X_1 0.11140846))

; Unscaled Input 2 : (-3.141592, -3.1315920000000004)
(assert (<= X_2 -0.498408347))
(assert (>= X_2 -0.499999896))

; Unscaled Input 3 : (900, 1200)
(assert (<= X_3 0.5))
(assert (>= X_3 0.227272727))

; Unscaled Input 4 : (600, 1200)
(assert (<= X_4 0.5))
(assert (>= X_4 0.0))

; unsafe if coc is  $\neg$  minimal
(assert (or
  (and (<= Y_1 Y_0))
  (and (<= Y_2 Y_0))
  (and (<= Y_3 Y_0))
  (and (<= Y_4 Y_0))
))

```

Limites

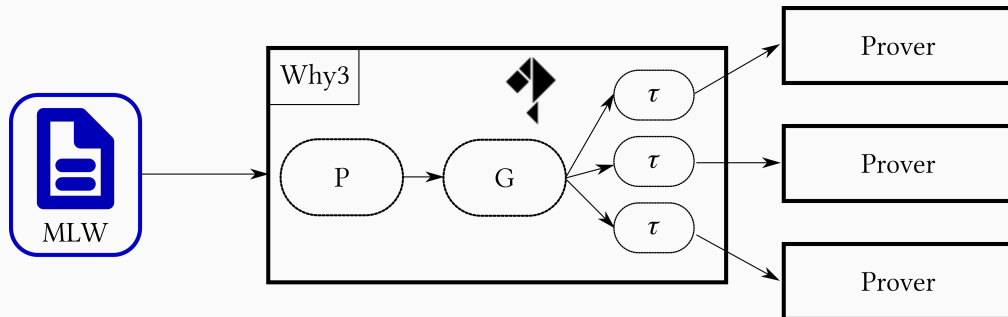
Exemple	Limite	Propriété désirable
$x \in \mathbb{R}^{50176}$	Taille de la spec linéaire en la taille des entrées?	<i>Compacité</i>
$\forall x \in E \subseteq \mathbb{R}^d, \epsilon \in \mathbb{R}, f_1(x)f_2(x) \leq \epsilon$	Spécifier des propriétés sur différents réseaux?	<i>Composabilité</i>
$\forall (x, x'), g(f, x, x') \in \mathcal{D}, f(x) = f(x')$	Comment définir et exprimer des fonctions? Des ensembles mathématiques?	<i>Expressivité</i>

Nos ingrédients pour l'instant

Plein de prouveurs état de l'art	Des modèles divers et variés	Un langage de spécification frustrant
Marabou [Kat+19], Z3 [dB08], CVC5 [Bar+22], Alt-Ergo [Con+18], PyRAT [LLG24], $\alpha - \beta$ -CROWN [Wan+21], Saver [RZ19], Plus du test métamorphique (Al- MOS [Lem+23])	Machine à vecteur de sup- port, réseau de neurone, arbres amplifiés par gra- dients	Type SMT-Lib avec de grosses restrictions sur les propriétés exprimables

Préparer une bonne vinaigrette avec Why3

Why3 à la rescousse



La plateforme de vérifications de programmes [FP13] Why3. Repose sur un langage de spécification et de programmation (WhyML), un parser (P), un générateur d'obligations de preuves (G), et des transformations (τ)

Spécification du langage

```
⟨decl⟩ ::= type ⟨tId⟩ = ⟨type⟩
        | predicate ⟨id⟩
          ⟨binder⟩* = ⟨expr⟩
        | function ⟨id⟩
          ⟨binder⟩* ⟨spec⟩* = ⟨expr⟩

⟨type⟩ ::= ⟨tId⟩
        | ⟨type⟩ → ⟨type⟩
        | (⟨type⟩, ..., ⟨type⟩)
        | vector ⟨type⟩
        | int | bool | float | string
        | model

⟨binder⟩ ::= ⟨id⟩ | (⟨id⟩ : ⟨type⟩)

⟨spec⟩ ::= requires {⟨expr⟩}
        | ensures {⟨expr⟩}

⟨bop⟩ ::= ≤ | ≥ | < | >
        | + | - | × | /
        | ^ | v | →
```

```
⟨expr⟩ ::= ⟨id⟩
        | ⟨built-in⟩
        | ⟨expr⟩ ⟨expr⟩
        | (⟨expr⟩, ..., ⟨expr⟩)
        | let ⟨id⟩ = ⟨expr⟩ in
        | if ⟨expr⟩ then ⟨expr⟩
          else ⟨expr⟩
        | ⟨expr⟩ ⟨bop⟩ ⟨expr⟩
        | forall ⟨binder⟩. ⟨expr⟩
        | exists ⟨binder⟩. ⟨expr⟩
        | not ⟨expr⟩
        | i ∈ Integer
        | {true, false} ∈ Boolean
        | f ∈ Float | s ∈ String
```

```
⟨built-in⟩ ::= read_model ⟨expr⟩
            | length ⟨expr⟩
            | has_length ⟨expr⟩ ⟨expr⟩
            | ⟨expr⟩[⟨expr⟩]
            | ⟨expr⟩@@⟨expr⟩
```

Grammaire pour le langage de CAISAR, très largement redevable de WhyML. Nos additions sont encadrées.

Exemple

theory ACASXU_P3

constant distance_to_intruder : int = 0;
[...]

predicate valid_input (i: input) =
 (0.0 :t) .≤ i[distance_to_intruder]
 .≤ (60760.0 :t)
 ∧ .- pi .≤ i[angle_to_intruder] .≤ pi
 ∧ .- pi .≤ i[intruder_heading] .≤ pi
 ∧ (100.0 :t) .≤ i[speed] .≤ (1200.0 :t)
 ∧ (0.0 :t) .≤ i[intruder_speed]
 .≤ (1200.0 :t)

constant nn_1_1: nn =
 read_neural_network "net.onnx" ONNX

predicate is_min (o: vector t) (i: int) =
 forall j: int. $0 \leq j < 5 \rightarrow i \neq j \rightarrow o[i] .\leq o[j]$

goal P3_1_1:
 forall i: vector t.
 has_length i 5 $\rightarrow$ valid_input i $\rightarrow$
 let is_coc_min = is_min (nn_1_1@@i) 0 **in**
 $\neg$ is_coc_min

end

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)



crédit : Randall Munroe

Comment mélanger la vinaigrette?

Exemple

Soient deux réseaux nn_1 et nn_2 , $x \in \mathbb{R}^2$, $\epsilon \in \mathbb{R} \ll 1$. Soit
 $r = nn_2(nn_1(x_1), x_1 + \epsilon) + nn_1(x_0)$. On souhaite garantir $r > 0$.

Exemple

Soient deux réseaux nn_1 et nn_2 , $x \in \mathbb{R}^2$, $\epsilon \in \mathbb{R} \ll 1$. Soit
 $r = nn_2(nn_1(x_1), x_1 + \epsilon) + nn_1(x_0)$. On souhaite garantir $r > 0$.

*Impossible d'exprimer cette propriété dans le langage d'entrée des gagnants de la
VNN-Comp 202X*

Exemple

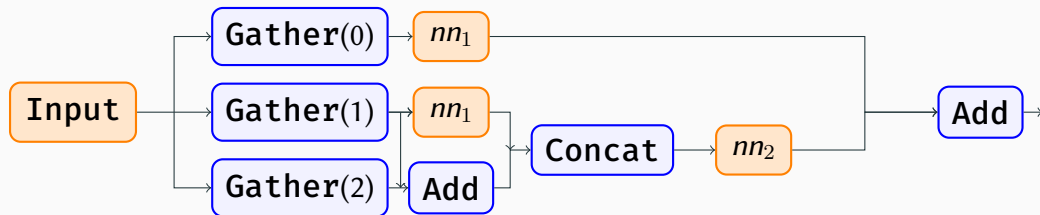
Transformer

$$\forall x_0, x_1, \epsilon. H(x_0, x_1, \epsilon) \rightarrow nn_2@@(nn_1@@(x_1), x_1 + \epsilon) + nn_1@@(x_0) > 0$$

en

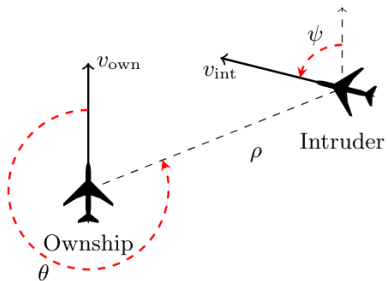
$$\forall x_0, x_1, \epsilon. H(x_0, x_1, \epsilon) \rightarrow nn_3@@(x_0, x_1, \epsilon) > 0$$

Exemple



Gather(0) (resp. **Gather(1)**) extrait x_0 (resp. x_1) et **Gather(2)** extrait ϵ depuis le nœud d'entrée.

Le premier nœud **Add** calcule $x_1 + \epsilon$. Les nœuds nn_1 nn_2 représentent les flots de contrôle de nn_1 et nn_2 inlinés. Enfin, **Concat** prépare les deux entrées requis par nn_2 .



Property ϕ_1 .

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- Tested on: all 45 networks.
- Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- Desired output property: the score for COC is at most 1500.

Property ϕ_2 .

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- Tested on: $N_{x,y}$ for all $x \geq 2$ and for all y .
- Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- Desired output property: the score for COC is not the maximal score.

Property ϕ_3 .

- Description: If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- Tested on: all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.
- Input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{\text{own}} \geq 980$, $v_{\text{int}} \geq 960$.
- Desired output property: the score for COC is not the minimal score.

Deuxième exemple : spécification non-normalisée ACAS-Xu.

Exemple (2)

```
let function normalize_t i mean range = (i .- mean) ./ range

let function denormalize_t i mean range = (i .* range) .+ mean

let function normalize_input i = Vector.mapi i normalize_by_index

let function denormalize_output_t o =
  denormalize_t o
  (7.51888402010059753166615337249822914600372314453125:t)
  (373.94992000000000200816430151462554931640625:t)

let runP1 (i: input) : t
  requires { has_length i 5 }
  requires { valid_input i }
  requires { intruder_distant_and_slow i }
  ensures { result .<= (1500.0:t) } =
    let j = normalize_input i in
    let o = (nn @@ j)[clear_of_conflict] in
    (denormalize_output_t o)
```

Figure 2 – Spécification pour la propriété ACAS ϕ_1 non normalisée.

Réécriture automatique de graphe

Intégration d'une partie de la spécification WhyML dans le flot de contrôle du réseau

1. support des hyperpropriétés (plusieurs réseaux/traces);
2. génération automatique d'un réseau intégrant la propriété;

Est-ce que la salade est digeste ?

Meh...

Property	Time (normalized)	Result (normalized)	Time (unnormalized)	Result (unnormalized)
ϕ_1	7.00	(✓)	9.00	(✓)
ϕ_2	11.00	(✓)	66.00	(✓)
ϕ_3	170.00	(✓)	444.00	(✓)
ϕ_4	26.00	(✓)	67.00	(✓)
ϕ_5	209.00	(✓)	576.00	(✓)
ϕ_6	72.00	(✓)	518.00	(✓)
ϕ_7	908.00	(⌚)	514.00	(ERR)
ϕ_8	908.00	(⌚)	624.00	(ERR)
ϕ_9	29.00	(X)	191.00	(X)
ϕ_{10}	6.00	(✓)	7.00	(✓)

Limitations existantes

- surcoût sur certains prouveurs ;
- arbitrage à trouver entre expressivité de la spécification, profondeur de la réécriture et support des prouveurs (peu de chances que les prouveurs supportent les conditionnelles)
- langage encore centré autour du programme : permettre la génération automatique de WhyML depuis des spécifications plus haut-niveau ?

Travaux en cours

1. explication formelles de décisions [WWB23];
2. intégration de notions de spécification conditionnelle [Ath+24]
3. vérification de propriétés LTL bornées;
4. support du *S-ONNX*;

Travaux en cours au sein du projet DeepGreen



Vérification de propriétés telles que l'iso-fonctionnalité entre réseaux sur plusieurs cibles matérielles ($nn_1(x) = nn_2(x)$), qualification de la perte de performance ($\forall x \in \mathcal{D}, \|nn_1(x) - nn_2(x)\| \leq \epsilon$), robustesse face aux transformations sémantiques



C A I S A R

Site web : <https://caisar-platform.com/>

Logiciel libre (LGPLv2) : <https://git.frama-c.com/pub/caisar>

Rapport technique : <https://hal.science/hal-03687211>

Offres : <https://caisar-platform.com/positions>

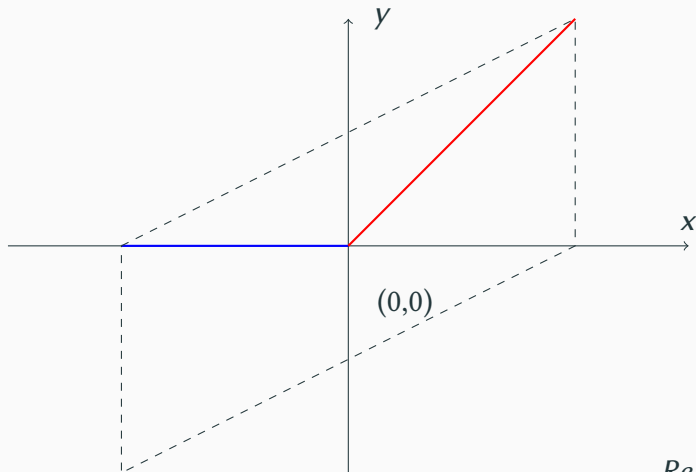
julien.girard2@cea.fr



DEEPGREEN

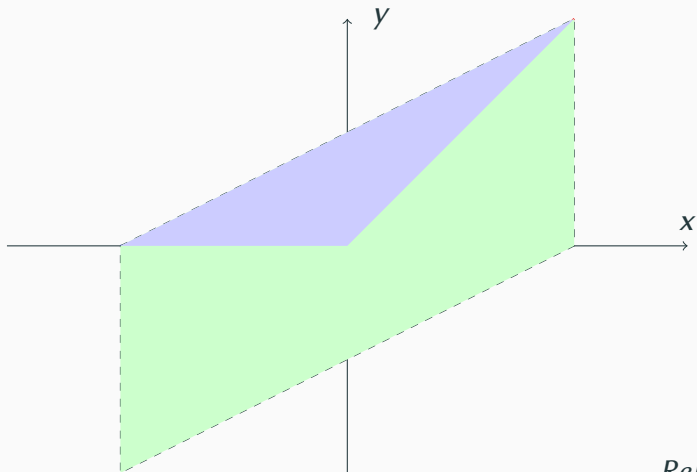
La maturation de CAISAR est partiellement financée par les projets PRISSMA, Confiance.ai et Deepgreen, ainsi que le projet ANR PEPR SAIF

La ReLU de tous les dangers



$$\text{ReLU}(x) = \max(x, 0)$$

La ReLU de tous les dangers



$$ReLU(x) = \max(x, 0)$$

Références

- [Ath+24] Anagha ATHAVALÉ et al. **“Verifying Global Two-Safety Properties in Neural Networks with Confidence”**. In : *Computer Aided Verification*. Sous la dir. d’Arie GURFINKEL et Vijay GANESH. ZSCC : 0000000 arXiv:2405.14400 [cs] type : article. Cham : Springer Nature Switzerland, juin 2024, p. 329-351. ISBN : 978-3-031-65630-9. DOI : 10.48550/arXiv.2405.14400. URL : <http://arxiv.org/abs/2405.14400> (visité le 23/07/2024) (cf. p. 34).

[Bak21] Stanley BAK. **“Nnenum : Verification of ReLU Neural Networks with Optimized Abstraction Refinement”**. In : *NASA Formal Methods*. Sous la dir. d’Aaron DUTLE et al. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2021, p. 19-36. ISBN : 978-3-030-76384-8. DOI : 10.1007/978-3-030-76384-8_2 (cf. p. 9-13).

- [Bar+22] Haniel BARBOSA et al. **“cvc5 : A Versatile and Industrial-Strength SMT Solver”**. In : *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*. Sous la dir. de Dana FISMAN et Grigore ROSU. T. 13243. Lecture Notes in Computer Science. Springer, 2022, p. 415-442. DOI : 10.1007/978-3-030-99524-9_24 (cf. p. 18).
- [Car+23] Nicholas CARLINI et al. **Extracting Training Data from Diffusion Models**. 2023. DOI : 10.48550/ARXIV.2301.13188. URL : <https://arxiv.org/abs/2301.13188> (cf. p. 8).

- [Con+18] Sylvain CONCHON et al. “**Alt-Ergo 2.2**”. In : *SMT Workshop : International Workshop on Satisfiability Modulo Theories*. Oxford, United Kingdom, juill. 2018. URL : <https://hal.inria.fr/hal-01960203> (cf. p. 18).
- [dB08] Leonardo DE MOURA et Nikolaj BJØRNER. “**Z3 : An Efficient SMT Solver**”. In : *Tools and Algorithms for the Construction and Analysis of Systems*. Sous la dir. de C. R. RAMAKRISHNAN et Jakob REHOF. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2008, p. 337-340. ISBN : 978-3-540-78800-3. DOI : 10.1007/978-3-540-78800-3_24 (cf. p. 18).

- [Dem+23] S. DEMARCHI et al. **“Supporting Standardization of Neural Networks Verification with VNN-LIB and CoCoNet.”**. In : *6th Workshop on Formal Methods for ML-Enabled Autonomous Systems*. Juill. 2023 (cf. p. 16).
- [Dut+17] Souradeep DUTTA et al. **“Output Range Analysis for Deep Neural Networks”**. In : *arXiv:1709.09130 [cs, stat]* (sept. 2017). arXiv : 1709.09130 [cs, stat] (cf. p. 9-13).
- [Ehl17] Ruediger EHLERS. **“Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks”**. In : *arXiv:1705.01320 [cs]* (mai 2017). arXiv : 1705.01320 [cs] (cf. p. 9-13).

- [FP13] Jean-Christophe FILLIÂTRE et Andrei PASKEVICH. “**Why3 - Where Programs Meet Provers**”. In : *Programming Languages and Systems*. Sous la dir. de Matthias FELLEISEN et Philippa GARDNER. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2013, p. 125-128. ISBN : 978-3-642-37036-6. DOI : 10.1007/978-3-642-37036-6_8 (cf. p. 20).
- [Kat+17] Guy KATZ et al. “**Reluplex : An Efficient SMT Solver for Verifying Deep Neural Networks**”. In : *Computer Aided Verification*. Springer International Publishing, 2017, p. 97-117. ISBN : 9783319633879. DOI : 10.1007/978-3-319-63387-9_5 (cf. p. 7, 8).

- [Kat+19] Guy KATZ et al. **“The Marabou Framework for Verification and Analysis of Deep Neural Networks”**. en. In : *Computer Aided Verification*. Sous la dir. d’Isil DILLIG et Serdar TASIRAN. T. 11561. Cham : Springer International Publishing, 2019, p. 443-452. ISBN : 978-3-030-25539-8 978-3-030-25540-4. DOI : 10.1007/978-3-030-25540-4_26. (Visité le 18/07/2019) (cf. p. 9-13, 18).
- [KFS21] Haitham KHEDR, James FERLEZ et Yasser SHOUKRY. **“PEREGRiNN : Penalized-Relaxation Greedy Neural Network Verifier”**. In : (2021). arXiv : 2006.10864 [cs.LG] (cf. p. 9-13).
- [Lem+23] Augustin LEMESLE et al. **“AIMOS : Metamorphic Testing of AI - AnIndustrial Application”**. In : *WAISE 2023* (2023) (cf. p. 18).

- [LLG24] Augustin LEMESLE, Julien LEHMANN et Tristan Le GALL. **Neural Network Verification with PyRAT**. 2024. DOI : 10.48550/ARXIV.2410.23903 (cf. p. 9-13, 18).
- [Mad+17] Aleksander MADRY et al. **“Towards Deep Learning Models Resistant to Adversarial Attacks”**. In : *arXiv:1706.06083 [cs, stat]* (juin 2017). arXiv : 1706.06083 [cs, stat] (cf. p. 8).
- [Mül+21] Christoph MÜLLER et al. **“Scaling Polyhedral Neural Network Verification on GPUs”**. In : *Proceedings of Machine Learning and Systems* 3 (2021) (cf. p. 9-13).

- [PWW20] Brandon PAULSEN, Jingbo WANG et Chao WANG. “**ReluDiff : differential verification of deep neural networks**”. In : *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. ACM, juin 2020. DOI : [10.1145/3377811.3380337](https://doi.org/10.1145/3377811.3380337). URL : <http://dx.doi.org/10.1145/3377811.3380337> (cf. p. 9-13).
- [RZ19] Francesco RANZATO et Marco ZANELLA. “**Robustness Verification of Support Vector Machines**”. In : *Static Analysis*. Sous la dir. de Bor-Yuh Evan CHANG. Cham : Springer International Publishing, 2019, p. 271-295. ISBN : 978-3-030-32304-2 (cf. p. 15, 18).

- [SED21] David SHRIVER, Sebastian ELBAUM et Matthew B. DWYER. **“DNNV : A Framework for Deep Neural Network Verification”**. In : *Computer Aided Verification*. Sous la dir. d’Alexandra SILVA et K. Rustan M. LEINO. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2021, p. 137-150. ISBN : 978-3-030-81685-8. DOI : 10.1007/978-3-030-81685-8_6 (cf. p. 9-13).
- [Sin+19] Gagandeep SINGH et al. **“Beyond the Single Neuron Convex Barrier for Neural Network Certification”**. In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. WALLACH et al. Curran Associates, Inc., 2019, p. 15098-15109. URL : <http://papers.nips.cc/paper/9646-beyond-the-single-neuron-convex-barrier-for-neural-network-certification.pdf> (visité le 27/07/2020) (cf. p. 9-13).

- [Str+22] Christopher A. STRONG et al. **ZoPE : A Fast Optimizer for ReLU Networks with Low-Dimensional Inputs**. 2022. arXiv : 2106.05325 [cs.LG] (cf. p. 9-13).
- [TXT19] Vincent TJENG, Kai XIAO et Russ TEDRAKE. **“Evaluating Robustness of Neural Networks with Mixed Integer Programming”**. In : International Conference on Learning Representations (ICLR). 2019. URL : <https://openreview.net/pdf?id=HyGIdiRqtm> (visité le 19/06/2019) (cf. p. 9-13).
- [Urb+19] Caterina URBAN et al. **“Perfectly Parallel Fairness Certification of Neural Networks”**. In : *arXiv:1912.02499 [cs]* (déc. 2019). arXiv : 1912.02499 [cs] (cf. p. 8-13).

- [Wan+21] Shiqi WANG et al. **Beta-CROWN : Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification.** 31 oct. 2021. arXiv : 2103.06624 [cs, stat]. URL : <http://arxiv.org/abs/2103.06624> (visité le 04/03/2022) (cf. p. 9-13, 18).
- [WWB23] Min WU, Haoze WU et Clark BARRETT. **VeriX : Towards Verified Explainability of Deep Neural Networks.** 2023. arXiv : 2212.01051 [cs.LG] (cf. p. 34).

[Yan+21] Xiaodong YANG et al. **“Reachability analysis of deep ReLU neural networks using facet-vertex incidence”**. In : *Proceedings of the 24th International Conference on Hybrid Systems : Computation and Control*. HSCC '21. Nashville, Tennessee : Association for Computing Machinery, 2021. ISBN : 9781450383394. DOI : 10.1145/3447928.3456650. URL : <https://doi.org/10.1145/3447928.3456650> (cf. p. 9-13).