Caractériser des propriétés de confiance d'IA avec Why3

Julien Girard-Satabin, Michele Alberti, François Bobot, Zakaria Chihani

Caractériser des propriétés de confiance d'IA avec Why3

Julien Girard-Satabin, Michele Alberti, François Bobot, and Zakaria Chihani

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
julien.girard2@cea.fr
firstname.lastname@cea.fr

Résumé

Nous proposons de faire la démonstration de CAISAR, un logiciel libre en développement au sein du laboratoire de sûreté et sécurité des logiciels du CEA LIST. CAISAR est une plateforme permettant de caractériser la sûreté des logiciels résultant d'un protocole d'apprentissage (réseaux de neurones, machines à vecteur de support, forêts aléatoires). CAISAR se base sur la plateforme de vérification Why3 pour fournir un langage typé de modélisation de problèmes, ainsi qu'un moteur de raisonnement logique éprouvé. Cette approche autorise par exemple la vérification de systèmes composés de plusieurs blocs d'IA, ainsi qu'une meilleure expressivité des propriétés à vérifier. CAISAR est disponible à https://git.frama-c.com/pub/caisar.

Proposition de présentation

Contexte

Ces cinq dernières années ont vu naître le champ scientifique de la vérification formelle de programmes appris par réseaux de neurones. Ce champ est prospère : plus de vingt outils ont été ainsi développés sur cette période [12, 13, 21, 22, 18, 19, 16, 1, 11, 6, 14, 20, 7]. Ces travaux utilisent différentes techniques (interprétation abstraite, calcul SMT, programmation par contrainte), et utilisent souvent des heuristiques spécialisées sur des propriétés très spécifiques ; le plus souvent une invariance de la sortie par rapport à une perturbation de l'entrée circonscrite à un voisinage bien défini. Enfin, ces outils sont parfois limités quant aux programmes qu'ils peuvent vérifier, se limitant exclusivement aux réseaux de neurones convolutifs. Toutefois, nous observons que des idées développées dans certains outils percolent dans d'autres : ainsi, une heuristique spécifique faisant appel à des multiplicateurs de Lagrange dans l'outil *OVAL* [1] s'est ensuite vue réutilisée par ailleurs dans αβ-*CROWN* [22] et *ERAN* [19].

L'évolution rapide de ce domaine peut se comprendre comme la recherche, l'évaluation et la sélection de bonnes techniques de vérification par la communauté. Toutefois, à mesure que les programmes appris se voient intégrés dans des systèmes industriels, les exigences de sûreté et de sécurité vont croissant. Comment vérifier un système composé de différents programmes d'IA hétérogènes (par exemple : enchaînement de réseaux et de machines à vecteur de support) ? Comment vérifier des propriétés de conformité par rapport à des exigences fournies ? Pour tenter de répondre à ces questions, nous avons développé une plateforme dédiée à la caractérisation de la robustesse et de la sûreté des systèmes à base d'intelligence artificielle, CAISAR [9].

Principes guidant la conception de CAISAR

L'objectif de CAISAR est de fournir une interface unifiée de modélisation de problème de vérification sur les programmes à base d'IA. Pour se faire, CAISAR se base sur la plateforme de vérification *Why3* [8] et son langage WhyML. CAISAR modélise les propriétés à vérifier via des prédicats WhyML.

1. https://github.com/oval-group/oval-bab

Une propriété classique dans la littérature [22, 18, 19, 3] est la robustesse locale de la prédiction par rapport à une perturbation locale, dont voici la définition. Soit un espace d'entrée $\mathcal{X}$ (qu'on peut restreindre sans perte de généralité à un sous-ensemble de $\mathbb{R}^d$). Soit un programme $f : \mathcal{X} \mapsto \mathbb{R}$ associant à un échantillon $x \in \mathcal{X}$ une valeur réelle (par exemple, dans un problème de classification d'images, le programme associe à chaque échantillon une classe). Soit $\varepsilon \in \mathbb{R}$, et soit d une distance (on utilise typiquement les normes l_p pour $p \geq 1$). Un programme est alors localement robuste si, étant donné un échantillon $x_0 \in \mathcal{X}$:

$$\forall x, d(x, x_0) < \varepsilon \implies f(x) = f(x_0)$$

La Figure 1 montre comment écrire un prédicat WhyML pour modéliser cette propriété à vérifier, avec la norme l_∞ comme fonction de distance. CAISAR, s'inspirant de *Why3*, propose une interface de modélisation unifiée avant de décharger des obligations de preuves à différents prouveurs automatiques.

Quelques fonctionnalités

La plateforme gère principalement les réseaux de neurones sous Open Neural eXchange Format (ONNX) [2]. Ce choix est motivé par l'interopérabilité observée entre différents cadriciels d'apprentissage automatique (PyTorch, TensorFlow, Keras, Scikit-Learn). Les machines à vecteur de support sont gérées par un format csv *ad-hoc*. Enfin, il est possible pour l'utilisateur de spécifier un jeu de données particulier sur lequel vérifier les propriétés, via un fichier csv contenant les valeurs de chaque point de donnée. Actuellement, seuls les jeux de données de problèmes de classification peuvent être spécifiés ainsi : il est nécessaire de fournir une annotation explicite de la classe attendue pour chaque point.

Elle dispose d'une représentation intermédiaire du réseau de neurone permettant une traduction directe du réseau sous forme de formule logique, extractible en SMTLIB [2] grâce à *Why3*. Cette représentation intermédiaire ouvre également la voie à différentes transformations pour rendre la vérification plus simple, telles que la partition des entrées en régions linéaires [10].

CAISAR gère différents outils de vérification :

1. le Support Vector Machine reachability analysis tool (*SAVer* [15]), spécialisé dans les machines à vecteur de support
2. tous les solveurs standards prenant en entrée du SMTLIB (dont *Alt-Ergo* [4] et *Z3* [5]) ; les solveurs spécialisés en réseaux de neurones tels que *nnenum* [1] supportent un format voisin, VNN-Lib [3]
3. le prouveur SMT *Marabou* [13] spécialisé dans le traitement des réseaux de neurones grâce à différentes optimisations d'algorithme de pivot et de parallélisme
4. un outil de test métamorphique pour les IA, *AIMOS*, développé en interne et pour l'instant en source fermée
5. l'analyseur abstrait *PyRAT*, également développé en interne https://pyrat-analyzer.com/

Déroulement de la démonstration

La démonstration consiste d'abord à présenter le contexte au sein duquel se place CAISAR, et à quel besoin il répond. Nous présenterons les outils qu'il supporte ; et la nature des programmes traités (type de programme, taille, composabilité). Nous procéderons également à un comparatif avec d'autres approches comparables à CAISAR, telles que DNNV [17] et le tout récent Vehicle-lang [4]. Nous ferons

2. https://onnx.ai/
3. http://www.vnnlib.org/
4. https://github.com/vehicle-lang/vehicle

ensuite une démonstration de la spécification d'un problème de validation sur plusieurs problèmes jouets, avec différentes propriétés, en détaillant quand il est possible de le faire les mécanismes sous-jacents de la génération d'obligations de preuves. Nous ouvrirons enfin sur de potentielles futures fonctionnalités de la plateforme.

```
theory DataSetClassification

  type features = array t
  type class = int

  type datum = (features, class)
  type label_ = int

  type datum = (features, label_)

  type dataset = {
    nb_features: int;
    nb_label_s: int;
    data: array datum
  }
  [...]
  type model = {
    nb_inputs: int;
    nb_outputs: int;
  }

  function predict: model → features → label_
  [...]

  predicate linfty_distance (a: features) (b: features) (eps: t) =
    a.length = b.length ∧
    forall i: int. 0 ≤ i < a.length → .- eps .< a[i] .- b[i] .< eps

  predicate correct_at (m: model) (d: datum) =
    let (x, y) = d in
    y = predict m x
end
```

FIGURE 1 – Exemple de prédicats définis dans la bibliothèque standard de CAISAR. La fonction predict spécifie l'application d'un modèle m sur des entrées x. Le prédicat linfty_distance définit la norme l_∞, et correct_at est un prédicat qui vérifie si la prédiction d'un programme de classification est conforme à la vérité-terrain.

Remerciements

Le développement de CAISAR est partiellement soutenu par le programme Confiance.ai [5], et le projet PRISSMA [6].

Références

[1] Stanley Bak. Nnenum : Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods*, Lecture Notes in Computer Science, pages 19–36, Cham, 2021. Springer International Publishing.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[3] Marco Casadio, Ekaterina Komendantskaya, Matthew L Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan Refaeli. Neural network robustness as a verification property : a principled case study. In *International Conference on Computer Aided Verification*, pages 219–231. Springer, 2022.

[4] Sylvain Conchon, Albin Coquereau, Mohamed Iguernlala, and Alain Mebsout. Alt-Ergo 2.2. In *SMT Workshop : International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, July 2018.

[5] Leonardo de Moura and Nikolaj Bjørner. Z3 : An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 337–340, Berlin, Heidelberg, 2008. Springer.

[6] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output Range Analysis for Deep Neural Networks. *arXiv:1709.09130 [cs, stat]*, September 2017.

[7] Ruediger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. *arXiv:1705.01320 [cs]*, May 2017.

[8] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - Where Programs Meet Provers. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 125–128, Berlin, Heidelberg, 2013. Springer.

[9] Julien Girard-Satabin, Michele Alberti, François Bobot, Zakaria Chihani, and Augustin Lemesle. CAISAR : A platform for Characterizing Artificial Intelligence Safety and Robustness. In *AISafety*, CEUR-Workshop Proceedings, Vienne, Austria, July 2022.

[10] Julien Girard-Satabin, Aymeric Varasse, Marc Schoenauer, Guillaume Charpiat, and Zakaria Chihani. DISCO : Division of input space into convex polytopes for neural network verification. *JFLA*, 2021.

[11] P Henriksen and A Lomuscio. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *24th European Conference on Artificial Intelligence - ECAI 2020*, page 8, Santiago de Compostela, Spain, 2020.

[12] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex : An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

[13] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, volume 11561, pages 443–452. Springer International Publishing, Cham, 2019.

[14] Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition.

[15] Francesco Ranzato and Marco Zanella. Robustness verification of support vector machines. In *International Static Analysis Symposium*, pages 271–295. Springer, 2019.

5. https://www.confiance.ai/
6. https://prissma.univ-gustave-eiffel.fr/

[16] Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness Verification for Transformers. In *International Conference on Learning Representations*, 2020.

[17] David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. DNNV : A Framework for Deep Neural Network Verification. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 137–150, Cham, 2021. Springer International Publishing.

[18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and Effective Robustness Certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10802–10813. Curran Associates, Inc., 2018.

[19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[20] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. Perfectly Parallel Fairness Certification of Neural Networks. *arXiv:1912.02499 [cs]*, December 2019.

[21] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient Formal Safety Analysis of Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6367–6377. Curran Associates, Inc., 2018.

[22] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-CROWN : Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification.