Automated Synthesis of Certified Neural Networks: Initial Results and Open Research Lines

Matteo Zavatteri[1,*], Davide Bresolin[1] and Nicolò Navarin[1]

[1]University of Padova, Italy

Abstract

Certifying machine learning systems has become more and more important, especially when they are deployed in safety critical domains. In this paper, we sum up the work in [1] that combines Deep Learning with Formal Methods for the automated synthesis of certified neural networks and we discuss current open research lines.

Keywords

Certified AI, neural network, hard constraints, synthesis, CEGIS, MILP, quadratic programming

1. Introduction

Applications of neural networks include fields where they act as action advisors and controllers for safety-critical systems, where safety is paramount, and errors can be extremely expensive or dangerous. When a neural network is deployed in a safety-critical system, it is of extreme importance to prove that it satisfies the desired properties in all possible scenarios and under all possible inputs. Formal verification provides algorithms and methodologies that can exhaustively explore the state space of a system to guarantee that the system itself respects the desired properties, and it is routinely applied in the design of hardware and software systems. More recently, the challenge of the certification of neural networks has received attention from the formal verification community: the annual competition VNN-COMP [2] and the standard format VNN-LIB [3] have been established to compare state-of-the-art neural network verification tools and share benchmarks and test cases. The interest by the industry is testified by the recent discussion on the use of formal methods for neural networks certification in aviation [4], and by legislative texts such as the AI act by the European Union [5], that will require for all AI systems categorized as *high-risk* to be assessed before being deployed.

Many of the current approaches to verify a neural network reduce the problem to finding a solution to a constraint satisfaction problem (CSP). Such a CSP consists of a set of constraints encoding the neural network and a set of constraints encoding the property, and it is typically formulated within the frameworks of Mixed Integer Linear Programming (MILP), Satisfiability Modulo Theories (SMT), or more generally Constraint Programming (CP). Some selected references of these (and other) formal verification methods can be found in [6]. Some tools such as RELUPLEX [7] and Marabou [8] provide specialized versions of the SIMPLEX method to boost scalability of the formal verification phase.

However, verification alone can only give a yes or no answer and cannot *synthesize* a neural network that is guaranteed to respect the property when the answer to the verification problem is negative.

A few attempts in the literature to deal with the synthesis of certified neural networks have been provided. Some of them are restricted to general monotonicity of the network [9, 10], whereas others are limited to constraining the output for Hierarchical Multi-label classification Problems (e.g., [11, 12]). Considering more general rules, some works [13, 14, 15, 16] propose to inject constraints in the loss function of a neural network in such a way that the training of the model should prefer solutions where the constraints are satisfied. [17] generates counterexamples (solving an optimization problem) that violate some constraints that are expressed in first-order logic (FOL) and include them in the training set

OVERLAY 24: 6th International Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis

*Corresponding author.

matteo.zavatteri@unipd.it (M. Zavatteri); davide.bresolin@unipd.it (D. Bresolin); nicolo.navarin@unipd.it (N. Navarin)

0000-0001-6696-2972 (M. Zavatteri); 0000-0003-2253-9878 (D. Bresolin); 0000-0002-4108-1754 (N. Navarin)

in NLP tasks. [18] expresses constraints as logical clauses directly incorporated in the network structure as an additional layer that adds trainable parameters (clause weights) that represent the satisfiability level of constraints, providing explainability to the model as well as increased generalization. Some other methods additionally provide guarantees on the considered constraints, even though they either pose limitations on the constraints [19, 15], or rely on an external component to enforce the desired properties at run time [20, 21]. Finally, a recent review of deep learning with logical constraints is provided in [22].

The problem of automatically generating a computational system that provably satisfies a given high-level specification has been studied in computer since the seminal works of Church [23]. One relevant methodology developed by this line of research is the Counter Example Guided Inductive Synthesis (CEGIS) workflow, initially proposed in [24] for the automated synthesis of programs. The CEGIS workflow has been recently applied to prove stability of neural controllers [25, 26] and to generate invariants for dynamical systems [27, 28]. While these works exploit a learning-verify loop similar to our approach, they start from an analytical description of the dynamical system, and they consider only the problem of learning a network that satisfies the desired properties. A framework that is closer to our methodology is described in [29], where the CEGIS loop maintains a list of points on which the network is known to produce a faulty result, and then finds a modification to the weights of the network that simultaneously corrects all of these points. The loop is repeated until the network is verified. While this work aims to minimize the changes needed to certify the network, as we do, it restricts itself to change only the last layer of the network in the modification phase, due to the computational complexity of the underlying optimization problem. In our setting we start from a dataset sampled from an unknown distribution, and we aim to preserve the accuracy of the network as much as possible, without adding arbitrary restriction on how we modify the network. To do so, we develop a data generation technique to augment the training data that turns out to be crucial to keep the final network as close as possible to the one trained on the unknown data-generating distribution.

2. Certification of ReLU Neural Networks

The work in [1] targets feed-forward neural networks with rectified linear unit activation functions (ReLU). The employed framework used to certify such neural networks is based on the Counter Example Guided Inductive Synthesis (CEGIS) loop shown in Figure 1, a framework that was historically introduced in [24] for the automated synthesis of programs. In general, CEGIS works by generating a candidate solution to the problem at hand (a program in [24], a neural network in our case) and then formally verifying whether the candidate solution respects the desired properties or not. If not, the formal verification phase generates one or more counterexamples that are used to generate a new, better, solution, until the candidate solution is proved to respect the required properties.

To encode neural networks we relied on a straightforward encoding into linear real arithmetic (LRA), a fragment of first order logic that consists of arbitrary quantified boolean formulae on linear constraints. Given a neural network N with n inputs represented as the vector $\mathbf{x} = (x_1, \ldots, x_n)$, and m outputs represented as the vector $\mathbf{y} = (y_1, \ldots, y_m)$, the encoding builds a formula $F_N(\mathbf{x}, \mathbf{y})$, where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are the only free variables, and such that $F_N(\mathbf{x}, \mathbf{y})$ is true if and only if $\mathbf{y} = N(\mathbf{x})$.

To encode the properties we considered a fragment of LRA in which each formula has the following form:

$$F_{pre}(\mathbf{x}) \Rightarrow F_{post}(\mathbf{x}, \mathbf{y}),$$

where $F_{pre}(\mathbf{x})$ is a quantifier-free LRA formula over input variables only, and $F_{post}(\mathbf{x}, \mathbf{y})$ is a quantifier-free LRA formula over both input and output variables. Clearly, a neural network N is *certified* for a property $P := F_{pre}(\mathbf{x}) \Rightarrow F_{post}(\mathbf{x}, \mathbf{y})$ if the formula

$$F_{N,P}(\mathbf{x}, \mathbf{y}) := (F_N(\mathbf{x}, \mathbf{y}) \wedge F_{pre}(\mathbf{x})) \Rightarrow F_{post}(\mathbf{x}, \mathbf{y})$$

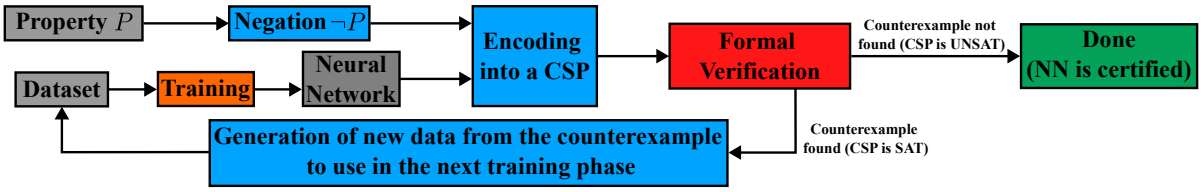

Figure 1: High level view of the CEGIS workflow employed for the synthesis of certified neural networks.

is valid[1]. This is the check that the formal verification phase in Figure 1 does. As usual, a formula is valid if and only if its negation is unsatisfiable. Hence, to check if N is certified for a property P, we can equivalently check whether $\neg F_{N,P}(\mathbf{x}, \mathbf{y})$ is unsatisfiable.

The output of the formal verification part can lead to two mutually-exclusive outcomes:

1. the CSP is unsatisfiable. In this case, no counterexample to P exists and N is certified for P;
2. the CSP is satisfiable. In this case there exists a counterexample $(\mathbf{x}, N(\mathbf{x}))$ such that $F_{N,P}(\mathbf{x}, N(\mathbf{x}))$ is false.

In the former case, the process is completed and we have proof that N respects the property P *for all possible values in the input domain*, and not only for the inputs contained in the (finite) dataset.

We now proceed to explain how to use CEGIS for repairing a neural network N in order to restore satisfaction of a violated property P. Let N be a ReLU neural network and P be a property on N. Suppose that $\neg F_{N,P}(\mathbf{x}, \mathbf{y})$ is satisfiable. To correct the behavior of the network, we want to build a pair $(\mathbf{x}, \mathbf{y}')$ such that $F_{post}(\mathbf{x}, \mathbf{y}')$ is true (notice that the inputs remain fixed as we can't control them). To do so, we start by building a new CSP, abstracted as an LRA formula F'_{post} built as a modification of F_{post} where:

- each x variable is replaced by its value assignment of the counterexample;
- each y variable is replaced by its value assignment of the counterexample plus a fresh variable s.

The result of this substitution is a new quantifier free LRA formula on s variables only (shifts that will be used to move y values in the counterexample to restore satisfiability of the property). If we solve the set of constraints represented by F'_{post} with respect to the objective function that minimizes the sum of squares of the various s, then we obtain a new value assignment $(\mathbf{x}, \mathbf{y}')$ that satisfies $F_{N,P}(\mathbf{x}, \mathbf{y})$, where each y' is equal to the value in the original counterexample plus the value of its related s obtained by solving the new CSP. This way we generate a new data point $(\mathbf{x}, \mathbf{y}')$ by selecting a $\mathbf{y}'$ that is at minimal distance from the original $\mathbf{y}$ and such that the property is respected. In fact, the purpose of the minimization problem is to generate an example as close as possible to the original data distribution while respecting property P. The pair $(\mathbf{x}, \mathbf{y}')$ is thus added to the current dataset (along with some other pairs sampled around and satisfying P), and a new iteration of the CEGIS loop starts.

In [1], we tested our framework to the MANNERS-DB example [30]. The case study involves a robot moving in a room where humans and animals are present. The controller is implemented through a neural network, where the outputs of the neural network provide numbers representing the adequacy of the possible actions executable by the robot. Each adequacy is a score of 1 to 5, where the bigger the number, the more adequate the action. The adequacy scores have been computed as the average across human judgments obtained via crowdsourcing (see [30]). We considered the following two properties:

P_1: *if the distance to the closest child is within 0.6 meters, then mopping the floor must be the less adequate action*;

P_2: *if the distance to the closest human is within 0.6 meters, then the adequateness of mopping the floor must be less than or equal to the adequateness of mopping the floor in all possible scenarios in which the distance to the closest animal is within 0.6 meters.*

[1]Note that the conjunct $F_N(\mathbf{x}, \mathbf{y})$ added on the left of the implication serves to exclude all pairs $(\mathbf{x}, \mathbf{y})$ such that $\mathbf{y} \neq N(\mathbf{x})$.

Notice that P_2 compares multiple (input, output) pairs. To encode it we extended our property language to consider two copies of the network $F_N(\mathbf{x}, \mathbf{y}) \wedge F_N(\mathbf{x}', \mathbf{y}')$ and allowing F_{pre} and F_{post} to be defined on both $\mathbf{x}, \mathbf{y}$ and $\mathbf{x}', \mathbf{y}'$, simultaneously.

We implemented our approach in Python 3, using PyTorch 2.1.1 as the deep learning framework and Gurobi 11 [31] to solve the various constraint satisfaction problems (see [32] for the source code). Gurobi can also produce multiple solutions to constraint satisfaction problems, a functionality that we exploited to generate multiple counterexamples. We run our software on a High Performance Computing (HPC) center using used a node equipped 2 Intel(R) Xeon(R) CPU E5-2650 v3 2.30GHz, 1 GPU Nvidia T4, 160GB of RAM, and running Ubuntu 20.04.3 LTS. The HPC is handled by the SLURM workload manager [33]. We assigned the corresponding job 20 CPU cores, 150GB of RAM, and the whole GPU of the node. With this settings, we could synthesize several certified neural networks with respect to different sizes of the hidden part (up to 10 hidden layers with 10 nodes each) and different learning rates (ranging from 10^{-2} to 10^{-4}). The fastest experiment took 2 CEGIS iterations and terminated in 43 seconds, whereas the longer experiment took 136 CEGIS iterations and terminated in about 14 hours and 15 minutes. Clearly, running the same experiments again will lead to different data due to the serveral randomness sources inside the whole process.

3. Open Research Lines

The work in [1] shed light on the following current limitations of the approach.

Computational complexity aspects. The neural networks we considered in [1] were fairly small. This choice was primarily made to keep the computational overhead reasonable in order to understand if the synthesis of certified neural networks was possible or not. The main bottlenecks that we identified are related to (i) training of the neural network, (ii) formal verification of the property, (iii) solving a QP problem to repair the counterexample, and (iv) total number of CEGIS iterations. Training scales with appropriate hardware that boosts parallelization (GPUs). Formal verification is in general NP-complete for this kind of networks and properties that fall under the class of constraints that we can handle [7]. A quadratic problem extends a MILP (which is NP-hard problem) by adding a quadratic objective function. At the moment the properties we are considering are simple, so this isn't a concern right now. Clearly, the bigger the F_{post} the worse the performances. The number of the CEGIS iterations depends on how good we are to augment the dataset with new (input,output) pairs that restore satisfiability of the property where it does not hold. This latter one is at the moment the most important, since every CEGIS iteration entails all other computationally expensive steps. To reduce the whole number of iterations, the training phase should inject soft constraints, to try to output a neural network which is already "close" to satisfy the property.

More expressive networks. The class of ReLU neural networks is straightforward to be encoded in LRA. However, in general we might consider neural networks with other non-linear activation functions (e.g., sigmoid) or a different class of constraints. Our approach is modular and thus can work with other networks and properties provided that we have methods and techniques to handle the corresponding CSP and repair problems. More non-linear functions can be handled by means of abstract interpretation techniques, where non-linearity is addressed by means of over-approximations. In this settings, if the formal verification phase does not find any counterexample, then we are sure that the network is certified. Instead, if counterexamples are found, we need to test whether they are spurious (i.e., not actual counterexamples) or not.

More expressive properties. At the moment we are verifying atemporal properties on the input-output pairs of the neural network. To assure that an actual control system behaves as expected, it is important to verify temporal properties on the control policy implemented by the neural network, to guarantee that the control objectives are achieved (liveness properties) in a safe manner (safety

properties). This would require to extend the property specification language to some temporal logic, as well as to lift the certification techniques to reinforcement learning algorithms. Another direction worth investigating is the synthesis of neural networks that are certified for more than one property. In any real scenario the network is expected to be certified on a large number of requirements.

Other case studies. We tested our approach on the MANNERS-DB example from [30]. Extending the experiments with datasets and examples coming from different domains is a worthwhile research direction. One test-case worth mentioning is the ACASXu (Airborne Collision Avoidance System X for unmanned aircrafts), where a set of neural networks act as action advisor for the possible aircraft's maneuvers, and that has become a de-facto benchmark for the verification of neural networks (see [7]). Another source of relevant test cases is the VNN-COMP [2].

4. Conclusions

This paper is a summary of [1] that proposes a method based on the CEGIS loop to automatically synthesize certified neural networks. Our approach is based on a formal verification part that seeks counterexamples that are then repaired and injected into training to correct wrong behavior of the network with respect to the properties to enforce.

Acknowledgments

This work was supported by the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.5 - Call for tender No. 3277 of 30 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU; project code: ECS00000043, Concession Decree No. 1058 of June 23, 2022, CUP C43C22000340006, project title "iNEST: Interconnected Nord-Est Innovation Ecosystem", and by Mission 4 Component 2 Investment 1.3 - Call for tender No. 341 of March 15, 2022 of Italian Ministry of University and Research – NextGenerationEU; project code PE0000013, Concession Decree No. 1555 of October 11, 2022, CUP C63C22000770006, project title "Future AI Research (FAIR) - Spoke 2 Integrative AI - Symbolic conditioning of Graph Generative Models (SymboliG)". This work was also supported by the Projects iNEST Young Researchers "Neuro-Symbolic AI in digital twins", and by the INdAM-GNCS 2024 Project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale" CUP_E53C23001670001.

References

[1] M. Zavatteri, D. Bresolin, N. Navarin, Automated synthesis of certified neural networks, in: ECAI 2024 - 27th European Conference on Artificial Intelligence, volume 392 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2024, pp. 1341–1348. doi:10.3233/FAIA240633.

[2] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, C. Liu, First three years of the international verification of neural networks competition (VNN-COMP), Int. J. Softw. Tools Technol. Transf. 25 (2023) 329–339.

[3] A. Tacchella, L. Pulina, D. Guidotti, S. Demarchi, VNN-LIB, https://www.vnnlib.org, 2022.

[4] EASA, Collins Aerospace, Formal Methods use for Learning Assurance (ForMuLA), Technical Report, 2023. URL: https://www.easa.europa.eu/en/downloads/137878/en.

[5] European Parliament, Artificial intelligence act, 2024. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206.

[6] A. Albarghouthi, Introduction to neural network verification, Found. Trends Program. Lang. 7 (2021) 1–157.

[7] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An efficient SMT solver for verifying deep neural networks, in: CAV 2017, LNCS, Springer, 2017, pp. 97–117.

[8] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, C. W. Barrett, The marabou framework for verification and analysis of deep neural networks, in: CAV 2019, volume 11561 of *LNCS*, Springer, 2019, pp. 443–452.
[9] X. Liu, X. Han, N. Zhang, Q. Liu, Certified monotonic neural networks, in: NeurIPS 2020, 2020.
[10] A. Sivaraman, G. Farnadi, T. D. Millstein, G. V. den Broeck, Counterexample-guided learning of monotonic neural networks, in: NeurIPS 2020, 2020.
[11] P. Dragone, S. Teso, A. Passerini, Neuro-symbolic constraint programming for structured prediction, in: (IJCLR 2021), volume 2986 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 6–14.
[12] E. Giunchiglia, T. Lukasiewicz, Multi-label classification neural networks with hard logical constraints, JAIR 72 (2021) 759–818.
[13] T. Li, V. Gupta, M. Mehta, V. Srikumar, A logic-driven framework for consistency of neural models, in: EMNLP-IJCNLP 2019, Association for Computational Linguistics, 2019, pp. 3922–3933.
[14] W. Wang, S. J. Pan, Integrating deep learning with logic fusion for information extraction, in: AAAI 2020, AAAI Press, 2020, pp. 9225–9232.
[15] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. V. den Broeck, A semantic loss function for deep learning with symbolic knowledge, in: ICML 2018, volume 80, PMLR, 2018, pp. 5498–5507.
[16] Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, H. Soh, Embedding symbolic knowledge into deep networks, in: NeurIPS 2019, 2019, pp. 4235–4245.
[17] P. Minervini, S. Riedel, Adversarially regularising neural NLI models to integrate logical background knowledge, in: CoNLL 2018, Association for Computational Linguistics, 2018, pp. 65–74.
[18] A. Daniele, L. Serafini, Knowledge enhanced neural networks for relational domains, in: AIxIA 2022, volume 13796 of *LNCS*, Springer, 2022, pp. 91–109.
[19] N. Hoernle, R. Karampatsis, V. Belle, K. Gal, Multiplexnet: Towards fully satisfied logical constraints in neural networks, in: AAAI 2022, AAAI Press, 2022, pp. 5700–5709.
[20] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu, Safe reinforcement learning via shielding, in: AAAI 2018, AAAI Press, 2018, pp. 2669–2678.
[21] C. Cornelio, J. Stuehmer, S. X. Hu, T. M. Hospedales, Learning where and when to reason in neuro-symbolic inference, in: ICLR 2023, OpenReview.net, 2023.
[22] E. Giunchiglia, M. C. Stoian, T. Lukasiewicz, Deep learning with logical constraints, in: IJCAI 2022, ijcai.org, 2022, pp. 5478–5485.
[23] A. Church, Logic, arithmetic, and automata, Journal of Symbolic Logic 29 (1964) 210–210.
[24] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, V. A. Saraswat, Combinatorial sketching for finite programs, in: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, ACM, 2006, pp. 404–415.
[25] A. Abate, D. Ahmed, M. Giacobbe, A. Peruffo, Formal synthesis of lyapunov neural networks, IEEE Control. Syst. Lett. 5 (2021) 773–778.
[26] Y. Chang, N. Roohi, S. Gao, Neural lyapunov control, in: NeurIPS, 2019, pp. 3240–3249.
[27] K. Chatterjee, T. A. Henzinger, M. Lechner, D. Zikelic, A learner-verifier framework for neural network controllers and certificates of stochastic systems, in: TACAS (1), Springer, 2023, pp. 3–25.
[28] A. Peruffo, D. Ahmed, A. Abate, Automated and formal synthesis of neural barrier certificates for dynamical models, in: TACAS (1), volume 12651 of *LNCS*, Springer, 2021, pp. 370–388.
[29] B. Goldberger, G. Katz, Y. Adi, J. Keshet, Minimal Modifications of Deep Neural Networks using Verification, in: LPAR 2023, volume 73 of *EPiC Series in Computing*, 2022, pp. 260–240.
[30] J. Tjomsland, S. Kalkan, H. Gunes, Mind your manners! A dataset and a continual learning approach for assessing social appropriateness of robot actions, Frontiers Robotics AI 9 (2022) 669420.
[31] Gurobi Optimization, LLC, Gurobi Reference Manual, 2024. URL: https://www.gurobi.com.
[32] M. Zavatteri, D. Bresolin, N. Navarin, Code for "automated synthesis of certified neural networks", 2024. URL: https://github.com/matteozavatteri/certified-nn-ecai24.
[33] M. A. Jette, T. Wickberg, Architecture of the slurm workload manager, in: Job Scheduling Strategies for Parallel Processing, Springer Nature Switzerland, Cham, 2023, pp. 3–23.