Formal Verification of Neural Networks?

Martin Leucker(✉)

Institute for Software Engineering and Programming Languages,
University of Lübeck, Lübeck, Germany
leucker@isp.uni-luebeck.de

Abstract. Machine learning is a popular tool for building state of the art software systems. It is more and more used also in safety critical areas. This demands for verification techniques ensuring the safety and security of machine learning based solutions. However, we argue that the popularity of machine learning comes from the fact that no formal specification exists which renders traditional verification inappropriate. Instead, validation is typically demanded, but formalization of so far informal requirements is necessary to give formal evidence that the right system is build. Moreover, we present a recent technique that allows to check certain properties for an underlying recurrent neural network and which may be uses as a tool to identify whether the system learned is right.

1 Introduction

When developing a program, one of the main fundamental problems in practice is to get it *correct*. Especially for safety-critical systems, lives may depend on the correct functioning of the software, for example, software controlling a car. But also in less safety-critical areas, the verification of systems plays an important economical role. Since its first days, it has been a challenge to get software systems correct, and a plethora of different methods have been developed ranging from debugging and testing to formal verification techniques. More generally, *formal methods* have been developed over the past decades to support the rigorous development of software systems. See [6] for an expert survey on formal methods, giving insight to the past, present, and future of formal methods.

On the other hand, artificial intelligence and as part of it machine learning is currently en vogue for developing software based systems. Especially deep learning methods turned out to be successful when developing applications for speech or image recognition, see for example [4,10,11], although it seems unclear, why such methods actually perform well in practice [14]. In general, deep learning methods may play one fundamental approach for synthesizing programs [7] rather than programming them manually.

As such, one may come up with the fundamental question of how to verify such networks for being sure that they adhere to what they are suppose to achieve. The latter question, however, has to be made precise. Let us first recall the common definition of verification:

G. Carvalho and V. Stolz (Eds.): SBMF 2020, LNCS 12475, pp. 3–7, 2020.
https://doi.org/10.1007/978-3-030-63882-5_1

Verification is a method for showing whether a system adheres to its specification.

Let us elaborate on this definition more carefully. The definition contains the notion of a *system* which is considered to be the system under test, the artefact that we are going to check. In our case it would be something like a (deep) neural network. Another ingredient of the definition is the notion of a *specification*. The specification defines the anticipated, correct behavior of the system to build. A huge variety of specification formalisms especially for specifying intended, correct behaviour have been developed in the past decade. A prominent example is Linear-time Temporal Logic (LTL), as suggested by Pnueli [13]. It has been noted that specifying the intended behaviour is inherently complex and approaches like specification patterns [5] or enriching temporal specification languages with syntactic sugar [3] have been developed to lower the burden of formulating specifications, just to name a two examples.

However, in machine learning applications (for which for example deep learning is employed), there is typically no specification on what the system is supposed to do. Instead, a neural network is *trained* on typical examples resulting in the final network.

We claim that it is crucial that machine learning works *without* a specification. We believe that its success—and its main application area—is in domains where a specification is ultimately complicated, nearly as complicated as programming a program itself. To illustrate our statement, think of specifying how a cat or a malignant tumor looks exactly. While it is easy for every human or a trained medical doctor to decide case by case whether a cat or a tumor is shown on a given image, respectively, a formalization seems extremely complicated.

So, in machine learning, we are facing the problem that we do not have a formal specification, that we only have examples of the intended behaviour of our system, but at the same time, we want assurance that the system works well in practice.

2 Verification vs. Validation vs. Trustworthy AI

Software engineering research has well understood the need for distinguishing the two questions:

- what a program is supposed to do according to its specification, and
- what a program is supposed to do according to the clients needs

where the client here reflects the stakeholder ordering a piece of software. While dealing with the first question is called *verification*, the methods for answering the second question are studied under the term *validation*. It is said that Barry Boehm memorably characterized the difference as follows[1]

[1] "In short, Boehm (3) expressed the difference between the software verification and software validation as follows: Verification: "Are we building the product right?" Validation: "Are we building the right product?" [12].

– Verification is building the system right.
– Validation is building the right system.

So, if at all, the quest for getting a safe system is first of all validation rather than verification. What does *right* mean, when the system is (only) given by means of examples but at the same time used in critical domains?

A quite elaborated notion that—as we claim—could be an interpretation of *right* is *trustworthy*. This term has been coined by a European expert group in the context of AI[2]. Rather than correctness, trustworthiness covers a broader understanding for building the system right. "'Trustworthy AI should be

1. lawful - respecting all applicable laws and regulations,
2. ethical - respecting ethical principles and values, and
3. robust - both from a technical perspective while taking into account its social environment."

To this extent, seven key requirements have been formulated that any trustworthy AI system should meet at least:[3]

– *Human agency and oversight* aiming for AI systems being human centric.
– *Technical Robustness and safety* aiming for AI systems to be resilient and secure, ensuring "that also unintentional harm can be minimized and prevented".
– *Privacy and data governance* including adequate data governance mechanisms.
– *Transparency* requesting that the data, system and AI business models should be transparent.
– *Diversity, non-discrimination and fairness* requiring that "unfair bias must be avoided".
– *Accountability* asking for measures ensuring responsibility and accountability.

The above key requirements are not exhaustive but give a clear perspective on what should be expected from the "right" AI-based system, also those based on neural networks.

As stated in [8], these requirements may be realized by technical and non-technical methods, and we consider it to be an important research topic to work on formal methods ensuring especially the above requirements.

3 Property-Directed Verification of Recurrent Neural Networks

Now, does the above discussion mean that formal verification such as model checking [2] is unimportant in the setting of AI and especially neural networks?

[2] https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai.

[3] The requirements are listed here in a very brief form. Please consult the original article for an elaborate explanation.

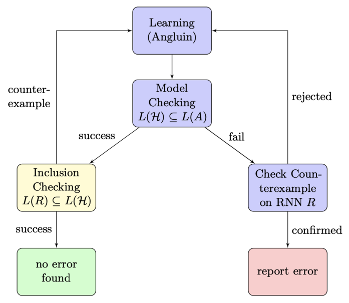
Learning (Angluin)
counter-example
rejected
Model Checking $L(\mathcal{H}) \subseteq L(A)$
success
fail
Inclusion Checking $L(R) \subseteq L(\mathcal{H})$
Check Counterexample on RNN R
success
confirmed
no error found
report error

Fig. 1. Property-directed verification of RNNs

We think that this is by no means the case but verification has to be understood differently. A (formal) specification may represent (a formalisation) of one of the requirements above and verification may be understood as a parameterized analysis algorithm showing a property of the underlying system. In simple words, research on verifying deep neural networks has to incorporate verification methods but especially formalization on what to check.

A recent approach for verifying properties of recurrent neural networks is given in [9]. It is observed that a recurrent neural network can be understood as an infinite-state machine, whenever it is used as a finite classifier. For such an infinite state system a finite-state automaton may be learned using automata learning techniques such as Angluin's L* [1] as a surrogate model approximating the system at hand. The surrogate model may then be used to check whether it meets its specification, for example using model checking techniques.

A surrogate model is a substitute model acting often in a specific role. [9] intertwines the task of verification and of learning the surrogate model in the form that a model is learned driven by the property to verify. While precise answers are obtained for checking the property on the *surrogate model* and an explicit counter example is provided if the underlying RNN does not satisfy the property at hand, a successful verification in terms of the *RNN* is only given up-to a given error probability. The latter is due to the fact that the infinite-state RNN is only statistically compared with the surrogate model. The procedure is sketched in Fig. 1, where the specification to check is denoted by a language $L(A)$ of an automaton, the RNN is denoted by R and the surrogate model to be learned is denoted by $\mathcal{H}$ as it acts as hypothesis in the setting of Angluin's L*.

[9] shows that the method is indeed beneficial in identifying both violations of a specification as well as the successful verification of properties (up-to a given error probability).

4 Conclusion

In this extended abstract we have discussed the role of verification in the setting of learning-based systems, which are derived by generalizing example behaviors.

We recalled that the success of such machine learning based approaches lies especially in application scenarios where no formal specification is given, when formalizing a specification is a challenge on its own.

We recalled key requirements developed by a European expert group on AI and posed their formalization and support by formal methods as a research agenda for forthcoming years.

Finally, we presented the gist of a novel verification approach for recurrent neural networks that is one example of an analysis algorithm rather than a verification algorithm parameterized by a given property. We suggest such algorithms as starting point for developing further formal methods for checking requirements on neural networks.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
2. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Bauer, A., Leucker, M.: The theory and practice of SALT. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 13–40. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20398-5_3
4. Deng, L., Hinton, G., Kingsbury, B.: New types of deep neural network learning for speech recognition and related applications: an overview. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8599–8603 (2013)
5. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) International Conference on Software Engineering, ICSE 1999, pp. 411–420. ACM (1999)
6. Garavel, H., Beek, M.H., Pol, J.: The 2020 expert survey on formal methods. In: ter Beek, M.H., Ničković, D. (eds.) FMICS 2020. LNCS, vol. 12327, pp. 3–69. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58298-2_1
7. Gulwani, S., Polozov, O., Singh, R.: Program synthesis. Found. Trends®in Program. Lang. 4(1–2), 1–119 (2017). https://doi.org/10.1561/2500000010
8. High-Level Expert Group on AI: Ethics guidelines for trustworthy AI. Report, European Commission, Brussels, April 2019. https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai
9. Khmelnitsky, I., et al.: Property-directed verification of recurrent neural networks (2020). Preprint https://arxiv.org/abs/2009.10610
10. Liu, J., et al.: Applications of deep learning to MRI images: a survey. Big Data Min. Anal. **1**(1), 1–18 (2018)
11. Mahmud, M., Kaiser, M.S., Hussain, A., Vassanelli, S.: Applications of deep learning and reinforcement learning to biological data. IEEE Trans. Neural Netw. Learn. Syst. **29**(6), 2063–2079 (2018)
12. Kopetz, H.: Automatic Error Detection. Software Reliability, pp. 68–80. Macmillan Education UK, London (1976). https://doi.org/10.1007/978-1-349-86129-3_9
13. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977), Providence, Rhode Island, 31 October–2 November 1977, pp. 46–57. IEEE Computer Society Press (1977)
14. Sejnowski, T.J.: The unreasonable effectiveness of deep learning in artificial intelligence. CoRR abs/2002.04806 (2020). https://arxiv.org/abs/2002.04806