Property-Directed Verification of Recurrent Neural Networks

Xuan Xie

`xuanxie@mpi-sws.org`
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Recurrent neural networks (RNNs) are a state-of-the-art tool to represent and learn sequence-based models. They have applications in time-series prediction, sentiment analysis, and many more. In particular, they are increasingly used in safety-critical applications and act, for example, as controllers in cyber-physical systems. Thus, there is a growing need for formal verification. However, research in this domain is only at the beginning. While formal-methods based techniques, such as model checking, have been successfully used in practice and reached a certain level of industrial acceptance, a transfer to machine-learning algorithms has yet to take place. We apply it on machine-learning artifacts rather than on the algorithm.

In this presentation, I will present a **property-directed approach to verifying recurrent neural networks (PDV)**. To this end, we learn a deterministic finite automaton as a surrogate model from a given RNN using *active automata learning*. This model may then be analyzed using *model checking* as a verification technique. The term *property-directed* reflects the idea that our procedure is guided and controlled by the given property rather than performing the two steps separately.

Property-directed verification (PDV). Let us give a glimpse of our method. We consider an RNN R as a binary classifier of finite sequences over a finite alphabet Σ. In other words, R represents the set of strings that are classified as positive. We denote this set by $L(R)$ and call it the *language* of R. Note that $L(R) \subseteq \Sigma^*$. We would like to know whether R is compatible with a given specification A, written $R \models A$. Here, we assume that A is given as a (deterministic) finite automaton. Finite automata are algorithmically feasible, albeit having a reasonable expressive power: many abstract specification languages such as temporal logics or regular expressions can be compiled into finite automata.

But what does $R \models A$ actually mean? In fact, there are various options. If A provides a complete characterization of the sequences that are to be classified as positive, then $\models$ refers to language equivalence, i.e., $L(R) = L(A)$. Note that this would imply that $L(R)$ is supposed to be a regular language, which may rarely be the case in practice. Therefore, we will focus on checking inclusion $L(R) \subseteq L(A)$, which is more versatile as we explain next.

Suppose N is a finite automaton representing a negative specification, i.e., R must classify words in $L(N)$ as negative at any cost. In other words, R does not produce false positives. This amounts to checking that $L(R) \subseteq L(\overline{N})$ where $\overline{N}$ is the "complement automaton" of N. For instance, assume that R is supposed to recognize valid XML documents, which, seen as a set of strings, is not a regular language. We may want to make sure that every opening tag is eventually followed by a corresponding closing tag[1], though the number of opening and the number of closing tags may differ. As negative specification, we can then take an automaton N accepting the corresponding *regular* set of strings. For example, `<book><author></author><author></book>` $\in L(N)$, since the second occurrence of `<author>` is not followed by some `</author>` anymore. On the other hand, we have `<book><author><author></author></book>` $\in L(\overline{N})$.

Symmetrically, suppose P is a finite automaton representing a *positive* specification so that we can find false negative classifications: If P represents the words that R *must classify as positive*, we would like to know whether $L(P) \subseteq L(R)$. Our procedure can be run using the complement of A as specification and inverting the outputs of R, i.e., we check, equivalently, $L(\overline{R}) \subseteq L(\overline{P})$. An important instance of this setting is *adversarial robustness certification*, which measures a neural network's resilience against adversarial examples. Given a (regular) set of words L classified as positive by the given RNN, the RNN is *robust* wrt. L if slight modifications in a word from L do not alter the RNN's judgement. This notion actually relies on a distance function. Then, P is the set of words whose distance to a word in L is bounded by a predefined threshold, which is regular for several popular distances such as the *Hamming* or *Levenshtein distance*. Similarly, we can also check whether the neighborhood of a regular set of words preserves a negative classification.

So, in all these cases, we are faced with the question of whether the language of an RNN R is contained in the (regular) language of a finite automaton A. Our approach to this problem relies on black-box checking [**?**], which has been designed as a combination of model checking and testing in order to verify finite-state systems and is based on Angluin's L* learning algorithm [1]. L* produces a sequel of *hypothesis* automata based on queries to R. Every such hypothesis $\mathcal{H}$ may already share some structural properties with R. So, instead of checking conformance of $\mathcal{H}$ with R, it is worthwhile to first check $L(\mathcal{H}) \subseteq L(A)$ using classical model-checking algorithms. If the answer is affirmative, we apply statistical model checking to check $L(R) \subseteq L(\mathcal{H})$ to confirm the result. Otherwise, a counterexample is exploited to refine $\mathcal{H}$, starting a new cycle in L*. Just like in black-box checking, our experimental results suggest that the process of interweaving automata learning and model checking is beneficial in the verification of RNNs and offers advantages over more obvious approaches such as (pure) statistical model checking or running automata extraction and model checking in sequence. A further key advantage of our approach is that, unlike in statistical model checking, we often

[1] over a finite predefined set of tags, which are contained in Σ

Table 1. Experimental results

Type	*Avg time* (s)	*Avg len*	*#Mistakes*	*Avg MQs*
SMC	92	111	**122**	286063
AAMC	444	**7**	30	3701916
PDV	**21**	11	109	**28318**

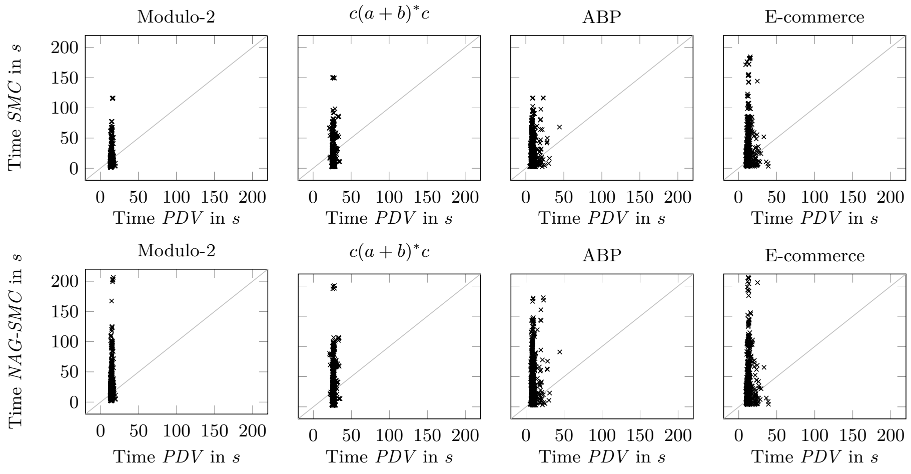

Fig. 1. Comparison of three algorithms on regular languages

find a *family* of counterexamples, in terms of loops in the hypothesis automaton, which testify conceptual problems of the given RNN.

Finally, I will demonstrate some comparisons of our algorithms against statistical model checking (SMC) and automaton abstraction model checking (AAMC) [2] on three applications, which are synthetic specifications, adversarial robustness certification and identifying contact sequences in contact tracing. Table 1 summarizes the executions of the three algorithms on the synthetic random specifications. Figure 1, which is a set of scatter plots, shows the results of the average time of executing PDV, SMC and neighbourhood automaton generation with statistical model checking (NAG-SMC) for certifying adversarial robustness on two random regular languages and two real-world datasets [3]. The x-axis and y-axis are both time in seconds, and each data point represents one adversarial robustness certification procedure. The execution time for PDV is constant, since, after successfully learning the surrogate model, PDV can efficiently certify the robustness. The experimental result shows the effectiveness and efficiency of our technique.

This is joint work with Igor Khmelnitsky, Daniel Neider, Rajarshi Roy, Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Martin Leucker and Lina Ye. It is currently under submission in ATVA 2021.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. 75(2), 87–106 (1987)
2. Khmelnitsky, I., Neider, D., Roy, R., Barbot, B., Bollig, B., Finkel, A., Haddad, S., Leucker, M., Ye, L.: Property-directed verification of recurrent neural networks. CoRR abs/2009.10610 (2020), `https://arxiv.org/abs/2009.10610`
3. Mayr, F., Yovine, S.: Regular inference on artificial neural networks. In: Proceedings of CD-MAKE 2018. Lecture Notes in Computer Science, vol. 11015, pp. 350–369. Springer (2018)