On-the-fly Black-Box Probably Approximately Correct Checking of Recurrent Neural Networks (Submitted version)*

Franz Mayr[1], Ramiro Visca[2], and Sergio Yovine[3]

[1] Universidad ORT Uruguay, `mayr@ort.edu.uy`
[2] Universidad ORT Uruguay, `visca@ort.edu.uy`
[3] Universidad ORT Uruguay, `yovine@ort.edu.uy`

Abstract. We propose a procedure for checking properties of recurrent neural networks without any access to their internal structure nor code. Our approach is a case of black-box checking based on learning a probably approximately correct, regular approximation of the intersection of the language of the black-box (the network) with the complement of the property to be checked, without explicitly building automata-based individual representations of them. When the algorithm returns an empty language, there is a proven upper bound on the probability of the network not verifying the requirement. When the returned language is nonempty, it is certain the network does not satisfy the property. In this case, a regular language approximating the intersection is output together with true sequences of the network violating the property. We show that this approach offers better guarantees than post-learning verification where the property is checked on a learned model of the network alone. Besides, it does not require resorting to an external decision procedure for verification nor fixing a specific requirement specification formalism.

1 Introduction

Today, deep recurrent neural networks (RNN), such as LSTM [13], are used to accomplish safety and security-critical tasks in a number of application areas, such as autonomous driving [18, 31], intrusion detection [17, 45], malware detection [28, 30, 39], and human activity recognition [33]. Therefore, there is increasing interest in verifying their behavior with respect to the requirements they must fulfill to correctly perform their task. However, accuracy-based metrics on test datasets do not provide provable probabilistic guarantees about the network outcomes [19].

For RNN devoted to sequence classification, one way of tackling this problem consists in learning a deterministic finite automaton (DFA) out of the network.

* Published in Machine Learning and Knowledge Extraction - International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25-28, 2020, Proceedings. Lecture Notes in Computer Science 12279, Springer 2020, ISBN 978-3-030-57320-1. Published: `https://link.springer.com/chapter/10.1007/978-3-030-57321-8_19`

Since RNN are more expressive than DFA [25], the language of the automaton is, in general, an approximation of the sequences classified as positive by the RNN. This approach can be implemented by resorting to algorithms such as the ones proposed in [21, 40, 42, 43]. Once the automaton is obtained, it can be model-checked against a desired property using an appropriate model-checker.

From a verification perspective, this general approach has several drawbacks, notably but not exclusively that the automaton learned from the RNN may be too large to be explicitly constructed. Moreover, the algorithms proposed in [40, 42, 43] do not provide quantitative assessments on how precisely the extracted automaton characterizes the language of RNN. Besides, these algorithms are white-box and they are tailored to specific classes of RNN. Thus, they cannot be used in contexts where privacy and security constraints prevent the network weights, structure, and/or code to be revealed.

Indeed, these issues are overcome by the black-box algorithm proposed in [21] which extends L^* to learn DFA which are probably correct approximations [37] of the RNN. However, the inconvenience still remains in finding real counterexamples on the network when the model-checker fails to verify the property on the model. To some extent, this complication can be dealt with learning-based black-box checking (BBC) [29]. BBC is a refinement procedure where finite automata are incrementally built and model-checked against a requirement. Counterexamples generated by the model-checker are validated on the black-box. False counterexamples are used to refine the automata. However, BBC requires fixing a formalism for specifying the requirements, typically linear-time temporal logic, and an external model-checker. Besides, the black-box is assumed to be some kind of finite-state machine.

To handle the problem of checking properties over RNN on a black-box setting without the downsides of BBC, we propose a method which performs on-the-fly checking during learning without resorting to an external model-checker. Our approach considers both the RNN and the property as black-boxes and it does not explicitly build nor assumes any kind of state-based representation of them. The key idea consists in learning a regular language which is a probably correct approximation of the intersection of the language of the RNN and the negation of the property. On one hand, when the returned language is empty, this approach ensures there is an upper bound on the probability of the network not satisfying the property. This bound is a function of the parameters of the algorithm. On the other, if the returned language is nonempty, the requirement is guaranteed to be false, and truly bad sequences of the RNN are provided together with a probably correct approximation of the language of faulty behaviors.[4]

Outline Section 2 briefly reviews probably approximately correct learning and defines the notion of on-the-fly property checking through learning. Section 3 revisits the problem of regular language learning and develops the main theoret-

[4] It is worth noticing that our approach differs from on-the-fly BBC as defined in [29] which relies on a strategy for seeking paths in the automaton of the requirement.

ical ideas. Section 4 experimentally validates the results. The other sections are devoted to related work and conclusions.

2 On-the-fly property checking through learning

2.1 PAC learning

We briefly revisit here *Probably Approximately Correct* (PAC) learning [37]. This summary is mostly based on [4].

Let $\mathcal{X}$ be the *universe* of examples. The *symmetric difference* of $X, X' \subset \mathcal{X}$, denoted $X \oplus X'$, is defined as $X \setminus X' \cup X' \setminus X$, where $X \setminus X'$ is $X \cap \overline{X'}$ and $\overline{X}$ is the complement of X. Examples are assumed to be identically and independently distributed according to an unknown probability distribution $\mathcal{D}$ over $\mathcal{X}$.

A *concept* C is a subset of $\mathcal{X}$. A concept class $\mathcal{C}$ is a set of concepts. Given an *unknown* concept $C \in \mathcal{C}$, the purpose of a *learning* algorithm is to output a hypothesis $H \in \mathcal{H}$ that *approximates* the target concept C, where $\mathcal{H}$, called *hypothesis space*, is a class of concepts possibly different from $\mathcal{C}$.

Approximation between concepts C and H is measured with respect to $\mathcal{D}$ as the probability of an example $x \in \mathcal{X}$ to be in their symmetric difference. This measure, also called *prediction error*, is formalized as $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \oplus H\right]$.

An *oracle* **EX**, which depends on C and $\mathcal{D}$, takes no input and draws i.i.d examples from $\mathcal{X}$ following $\mathcal{D}$, and tags them as *positive* or *negative* according to whether they belong to C or not. Calls to **EX** are independent of each other.

A PAC-learning algorithm takes as input an *approximation* parameter $\epsilon \in (0, 1)$, a *confidence* parameter $\delta \in (0, 1)$, a *target* concept $C \in \mathcal{C}$, an oracle **EX**, and a hypothesis space $\mathcal{H}$, and if it terminates, it outputs an $H \in \mathcal{H}$ which satisfies $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \oplus H\right] \leq \epsilon$ with confidence at least $1 - \delta$, for any $\mathcal{D}$. The output hypothesis H is said to be an ϵ-approximation of C with confidence $1-\delta$. Hereinafter, we refer to H as an (ϵ, δ)-approximation.

The concept class $\mathcal{C}$ is said to be *learnable* in terms of $\mathcal{H}$ if there exists a PAC-learning algorithm that terminates in polynomial time, measured in terms of its relevant parameters ϵ, δ, the size of the representations of examples and concepts, and the number of examples generated by **EX**.

PAC-learning algorithms may be equipped with other oracles. In this paper, we consider algorithms that make use of *membership* and *equivalence* query oracles, denoted **MQ** and **EQ**, respectively. **MQ** takes as input an example $x \in \mathcal{X}$ and returns whether $x \in C$ or not. **EQ** takes as input a hypothesis H and answers whether H is an (ϵ, δ)-approximation of C by drawing a sample $S \subset \mathcal{X}$ using **EX**, and checking whether for all $x \in S$, $x \in C$ iff $x \in H$, or equivalently, $S \cap (C \oplus H) = \emptyset$. The size m_S of S must be large enough to ensure the PAC criterion. Actually, m_S must be larger than $\frac{1}{\epsilon}\log\frac{1}{\delta}$ [12].

Lemma 1. *Let H be an (ϵ, δ)-approximation of C. For any $X \subseteq C \oplus H$, we have that $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in X\right] \leq \epsilon$ with confidence $1 - \delta$.*

Proof. For any $X \subseteq C \oplus H$, we have that $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H]$. Then, $\mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H] \leq \epsilon$ implies $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \epsilon$. Now, for any $S \subset \mathcal{X}$ such that $S \cap (C \oplus H) = \emptyset$, it follows that $S \cap X = \emptyset$. Therefore, any sample drawn by **EQ** that ensures $\mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H] \leq \epsilon$ with confidence $1-\delta$ also guarantees $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \epsilon$ with confidence $1-\delta$. □

Proposition 1. *Let H be an (ϵ, δ)-approximation of C. For any $X \subseteq \mathcal{X}$:*

$$\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \cap \overline{H} \cap X\right] \leq \epsilon \tag{1}$$

$$\mathbb{P}_{x\sim\mathcal{D}}\left[x \in \overline{C} \cap H \cap X\right] \leq \epsilon \tag{2}$$

with confidence at least $1-\delta$.

Proof. From Lemma 1 because $C \cap \overline{H} \cap X$ and $\overline{C} \cap H \cap X$ are subsets of $C \oplus H$. □

We will make use of Proposition 1 in the context of property checking.

2.2 Using learning for property checking

Let us suppose we want to verify whether $C \subseteq P$, equivalently $C \cap \overline{P} = \emptyset$, for some concept $C \in \mathcal{C}$ and for some property $P \subset \mathcal{X}$, such that we do not have a verification procedure for it.

2.2.1 Post-learning verification Let us assume we have a procedure for checking emptiness in $\mathcal{H}$ and $\overline{P} \in \mathcal{H}$. In this case, we can pose the problem as checking whether $H \cap \overline{P} = \emptyset$, where H is a PAC-learned model of C.

Proposition 2. *Let H be an (ϵ, δ)-approximation of C. For any $\overline{P} \in \mathcal{H}$:*

1. *if $H \cap \overline{P} = \emptyset$ then $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \cap \overline{P}\right] \leq \epsilon$, and*
2. *if $H \cap \overline{P} \neq \emptyset$ then $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in \overline{C} \cap H \cap \overline{P}\right] \leq \epsilon$,*

with confidence at least $1-\delta$.

Proof.
1. If $H \cap \overline{P} = \emptyset$ then $C \cap \overline{P} = C \cap \overline{H} \cap \overline{P}$. Thus, from Proposition 1(1) we have that $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \cap \overline{H} \cap \overline{P}\right] \leq \epsilon$, with confidence at least $1-\delta$.
2. If $H \cap \overline{P} \neq \emptyset$, from Proposition 1(2) we have that $\mathbb{P}_{x\sim\mathcal{D}}\left[x \in \overline{C} \cap H \cap \overline{P}\right] \leq \epsilon$, with confidence at least $1-\delta$. □

In words, whichever the outcome of the verification procedure for H, the probability of this verdict not holding for C is bounded by the approximation parameter ϵ, with confidence at least $1-\delta$.

Remark 1. *This approach has an important drawback. When $H \cap \overline{P} \neq \emptyset$, even if with small probability, counterexamples found by the verification procedure may not be in C. Therefore, whenever that happens, we would need to make use of the oracle* **EX** *to draw examples from $H \cap \overline{P}$ and tag them as belonging to C or not in order to trying finding a concrete counterexample in C. Moreover,* **EX** *may not be available at this time as it is part of the PAC-learning process but not the necessarily of the model-checking one.*

2.2.2 On-the-fly property checking through learning Rather than learning an (ϵ, δ)-approximation of C, an alternative is to use the PAC-learning algorithm to learn an (ϵ, δ)-approximation of $C \cap \overline{P} \in \mathcal{C}$.

Proposition 3. *Let H be an (ϵ, δ)-approximation of $C \cap \overline{P} \in \mathcal{C}$. Then:*

1. *if $H = \emptyset$ then $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in C \cap \overline{P}\right] \leq \epsilon$, and*
2. *if $H \neq \emptyset$ then $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in H \setminus (C \cap \overline{P})\right] \leq \epsilon$,*

with confidence at least $1 - \delta$.

Proof. Straightforward from the fact that $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in (C \cap \overline{P}) \oplus H\right] \leq \epsilon$, with confidence at least $1 - \delta$. □

Therefore, if the learned hypothesis is the empty set, this approach yields the same probabilistic assurance of the property being true in C as in the post-learning verification one. Nevertheless, on-the-fly property checking through learning has several advantages:

- A model of the *target* concept C is not explicitly built.
- The property $\overline{P}$ does not need to be in the hypothesis space $\mathcal{H}$.
- If the result is not empty, it may be the case the oracle **EX** does actually generate an example $x \in C \cap \overline{P}$ during the run of the PAC-learning algorithm. In this case, x serves as a witness of the violation.

Hereinafter, we exploit this idea in the context of automata-theoretic property checking over RNN.

3 Verification of properties over languages

In this section, we consider the case where the universe $\mathcal{X}$ is the set of words Σ^* over a set of symbols Σ, the target concept is a language $C \subseteq \Sigma^*$, and the hypothesis class $\mathcal{H}$ is the set of *regular languages*. Such languages, represented as *deterministic finite automata* (DFA), can be learned with L^* [3].

Given $C, P \subseteq \Sigma^*$, the verification problem is posed as checking whether $C \cap \overline{P} = \emptyset$. The approach consists in learning a regular language which is an (ϵ, δ)-approximation of $C \cap \overline{P}$.

Remark 2. *It is important to notice that C, P, and $\overline{P}$ need not be regular. Hereinafter, regularity is not assumed except otherwise stated.*

For the sake of simplicity, we refer to a regular language or its DFA representation indistinctly. For instance, we write $A = \emptyset$ and $A \neq \emptyset$ to mean the language of A is empty and nonempty, respectively.

3.1 L^*

L^* is an iterative learning algorithm that constructs a DFA by interacting with a teacher which makes use of oracles **MQ** and **EQ**. Here, we only consider the PAC-based version of L^*. The learner builds a table of observations OT by interacting with the teacher. This table is used to keep track of which words are and are not accepted by the target language. OT is built iteratively by asking the teacher membership queries through **MQ**.

Algorithm 1 shows L^* pseudocode. OT is a finite matrix $\Sigma^* \times \Sigma^* \to \{0,1\}$. Its rows are split in two. The 'upper' rows represent a prefix-closed set words and the 'lower' rows correspond to the concatenation of the words in the upper part with every $\sigma \in \Sigma$. Columns represent a suffix-closed set of words. Each cell represents the membership relationship, that is, $OT[u][v] = \mathbf{MQ}(uv)$.

Algorithm 1: L^*

```
   Input  : $\epsilon$, $\delta$
   Output: DFA $A$
1  Initialize;
2  $i \leftarrow 0$;
3  repeat
4      $i \leftarrow i + 1$;
5      while $OT$ is not closed or not consistent do
6          if $OT$ is not closed then
7              $OT \leftarrow$ Close($OT$);
8          end
9          if $OT$ is not consistent then
10             $OT \leftarrow$ Consistent($OT$);
11         end
12     end
13     $A \leftarrow$ BuildAutomaton($OT$);
14     Answer $\leftarrow$ $\mathbf{EQ}(A, i, \epsilon, \delta)$;
15     if $Answer \neq Yes$ then
16         $OT \leftarrow$ Update($OT$, $Answer$);
17     end
18 until $Answer = Yes$;
19 return $A$;
```

L^* initializes OT_0 (line 1) with a single upper row $OT_0[\lambda]$, a lower row $OT_0[\sigma]$ for every $\sigma \in \Sigma$, and a single column for λ, with values $OT_0[u][\lambda] = \mathbf{MQ}(u)$.

At each iteration $i > 0$, L^* makes OT_i *closed* (line 7) and *consistent* (line 10). OT_i is closed if, for every row in the bottom part of the table, there is an equal row in the top part. OT_i is consistent if for every pair of rows u, v in the top part, for every $\sigma \in \Sigma$, if $OT_i[u] = OT_i[v]$ then $OT_i[u\sigma] = OT_i[v\sigma]$.

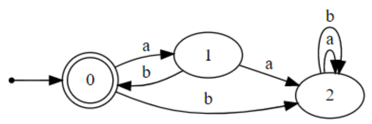
Once the table is closed and consistent, the algorithm proceeds to build the conjectured DFA A_i (line 13) which accepting states correspond to the entries of OT_i such that $OT_i[u][\lambda] = 1$.

Then, L^* calls **EQ** (line 14) to check whether A_i is PAC-equivalent to the target language. For doing this, **EQ** draws a sample S_i of size:

$$m_{S_i}(i,\epsilon,\delta) = \left\lceil \frac{i}{\epsilon} \log \frac{2}{\delta} \right\rceil \tag{3}$$

If for every $s \in S_i$, s belongs to the target language if and only if it belongs to the hypothesis A_i, the equivalence test is passed. In this case, L^* terminates and returns A_i. Otherwise, the learner receives a counterexample which violates the test and uses it to update OT (line 16). Then, it performs a new iteration.

Figure 1 shows an example of a run of L^* with target language $(ab)^*$.



(a)

OT_0	λ
λ	1
a	0
b	0

(b)

OT_1	λ
λ	1
a	0
b	0
aa	0
ab	1

(c)

OT_2	λ	b
λ	1	0
a	0	1
b	0	0
bb	0	0
aa	0	0
ab	1	0
ba	0	0
bba	0	0
bbb	0	0

(d)

Fig. 1. L^* example run.

Property 1 (From [3])**.** *If L^* terminates, it outputs an (ϵ,δ)-approximation of the target language. L^* always terminates if the target language is regular.*

Indeed, from Eq. 3, it follows that L^* increments the sample size at each iteration, making it larger than the lower bound $\frac{1}{\epsilon}\log\frac{1}{\delta}$. Therefore, the actual approximation error of a particular output of L^* satisfies the following property.

Lemma 2. *If L^* terminates with an* **EQ** *over a sample of size m, then its output is an $(\hat{\epsilon},\delta)$-approximation of the target language, for every $\hat{\epsilon} > \epsilon^\star(m,\delta)$, where*

$$\epsilon^\star(m,\delta) = \frac{1}{m}\log\frac{1}{\delta} \tag{4}$$

Proof. By Eq. 3, $m = m_{S_\ell}(\ell,\hat{\epsilon},\delta) > \frac{1}{\hat{\epsilon}}\log\frac{1}{\delta}$, where ℓ is the last iteration of L^*. Thus, $\epsilon^*(m,\delta)$ is the infimum of the approximation bounds assured by the output hypothesis, with confidence at least $1-\delta$. □

Lemma 3. *For every $i > 1$, if $A_i \neq \emptyset$ then the target language is nonempty.*

Proof. If $A_i \neq \emptyset$, there exists at least one accepting state, that is, there exists $u \in \Sigma^*$ such that $OT_i[u][\lambda] = 1$. Therefore, at some iteration $j \in [1, i]$, there is a positive membership query for u, i.e, $\mathbf{MQ}_j(u) = 1$. Hence, u belongs to the target language. □

Corollary 1. *If L^* returns a nonempty DFA for the target language $C \cap \overline{P}$, then $C \cap \overline{P} \neq \emptyset$. Moreover, every $u \in \Sigma^*$ such that $OT[u][\lambda] = 1$ is a counterexample.*

Proof. Immediate from Lemma 3. □

Proposition 4. *If L^* for $C \cap \overline{P}$ terminates with an* **EQ** *over a sample of size m and output a DFA A, then:*

1. *A is an $(\hat{\epsilon}, \delta)$-approximation of $C \cap \overline{P}$, for every $\hat{\epsilon} > \epsilon^\star(m, \delta)$.*
2. *If $A \neq \emptyset$ then $C \cap \overline{P} \neq \emptyset$.*

Proof. From Lemma 2, Corollary 1, and Proposition 3. □

Remark 3. *It is worth noticing that, by Lemma 3, from a verification proint of view, the execution of L^* for $C \cap \overline{P}$ could just be stopped as soon as the observation table has a non-zero entry.*

3.2 Bounded-L^*

When applied to learning regular approximations of concepts belonging to a more expressive class of languages, L^* may not terminate. In particular, this situation arrives when using this approach for learning languages of recurrent neural networks (RNN), since in general, this class of networks is strictly more expressive than DFA [25, 32, 34].

To cope with this issue, Bounded-L^* (Algorithm 2) has been proposed in [21]. It bounds the number of iterations of L^* by constraining the maximum number of states of the automaton to be learned and the maximum length of the words used to calling **EX**, which are typically used as parameters to determine the complexity of a PAC-learning algorithm [11].

Algorithm 2: Bounded-L^*

```
   Input  : MaxQueryLength, MaxStates, $\epsilon$, $\delta$
   Output: DFA $A$
1  Initialize;
2  $i \leftarrow 0$;
3  repeat
4      $i \leftarrow i + 1$;
5      while $OT$ is not closed or not consistent do
6          if $OT$ is not closed then
7              $OT$, QueryLengthExceeded $\leftarrow$ Close($OT$);
8          end
9          if $OT$ is not consistent then
10             $OT$, QueryLengthExceeded $\leftarrow$ Consistent($OT$);
11         end
12     end
13     if not QueryLengthExceeded then
14         $A \leftarrow$ BuildAutomaton($OT$);
15         Answer $\leftarrow$ EQ($A$, $i$, $\epsilon$, $\delta$);
16         MaxStatesExceeded $\leftarrow$ STATES($A$) $>$ MaxStates;
17         if Answer $\neq$ Yes and not MaxStatesExceeded then
18             $OT \leftarrow$ Update($OT$, Answer);
19         end
20     end
21     BoundReached $\leftarrow$ QueryLengthExceeded or MaxStatesExceeded;
22 until Answer = Yes or BoundReached;
23 return $A$;
```

The following property is a direct consequence of Property 1 of L^*.

Property 2 (From [21]). *If Bounded-L^* terminates with an automaton A which passes the* **EQ** *test, A is an (ϵ, δ)-approximation of the target language.*

However, upon termination Bounded-L^* may output an automaton A which fails to pass the **EQ** test, that is, A and the target language eventually disagree in $k > 0$ sequences of the sample S drawn by **EQ**. In such cases, the approximation bound guaranteed by the hypotheses produced by Bounded-L^* is given by the following property, which subsumes the previous property (case $k = 0$).

Property 3 (From [21]). *Assume Bounded-L^* terminates with an automaton A producing $k \in [0, m]$* **EQ***-divergences on a sample of size m, computed as in Eq. (3). Then, A is an $(\hat{\epsilon}, \delta)$-approximation of the target language, for every $\hat{\epsilon} > \epsilon^*$, where*

$$\epsilon^*(m, k, \delta) = \frac{1}{m-k} \log \frac{\binom{m}{k}}{\delta} \tag{5}$$

That is, $\epsilon^*(m, k, \delta)$ is the infimum of the approximation bounds assured by the output hypothesis, with confidence at least $1-\delta$, provided $k \geq 0$ divergences with the target language are found on a sample S of size m drawn by **EQ**.

A DFA output by Bounded-L^* can be used for post-learning verification. Clearly, Proposition 2 holds for any approximation parameter $\hat{\epsilon} > \epsilon^*(m, k, \delta)$, where $k \in [0, m]$ is the number of **EQ**-divergences upon termination (on a sample of size m).

On the other hand, if Bounded-L^* is used for on-the-fly verification, the theoretical guarantees are much more satisfactory indeed.

Theorem 1 (Main result). *If Bounded-L^* terminates with an automaton A producing $k \in [0, m]$* **EQ**-*divergences on a sample of size m for target $C \cap \overline{P}$, then:*

1. *A is an $(\hat{\epsilon}, \delta)$-approximation of $C \cap \overline{P}$, for every $\hat{\epsilon} > \epsilon^\star(m, k, \delta)$.*
2. *If $A \neq \emptyset$ or A does not pass the* **EQ** *test, then $C \cap \overline{P} \neq \emptyset$.*

Proof.
1. It follows directly from Property 3 and Proposition 4.
2. There are two cases. *a)* If $A \neq \emptyset$, then $C \cap \overline{P} \neq \emptyset$ by Proposition 4. *b)* If $A = \emptyset$ and A does not pass the **EQ** test, then $\emptyset \neq A \oplus (C \cap \overline{P}) = \emptyset \oplus (C \cap \overline{P}) = C \cap \overline{P}$. $\square$

Remark 4. *As for L^*, by Lemma 3, the execution of Bounded-L^* for $C \cap \overline{P}$ could just be stopped as soon as OT has a non-zero entry.*

4 Experimental results

We implemented a prototype of the proposed black-box on-the-fly property checking through learning based on Bounded-L^*. The approach consists in giving C and P as inputs to the teacher which serves as a proxy of $C \cap \overline{P}$. It is important to emphasize that this approach does not require modeling $\overline{P}$ in any particular way. Instead, to answer $\mathbf{MQ}(u)$ on a word u, the teacher evaluates $P(u)$, complements the output and evaluates the conjunction with the output of $C(u)$. To answer $\mathbf{EQ}(H)$, it draws a sample S of the appropriate size and evaluates $\mathbf{MQ}(u) \iff H(u)$ on every $u \in S$. Therefore, P may be any kind of property, even not regular, or another RNN.

We carried out three kinds of experiments. First, we studied RNN trained with sequences generated by Dyck context-free grammars (CFG) and checked regular and non-regular properties. Second, we check regular properties over RNN trained with sequences of software systems modeled as DFA. Third, we studied domain-specific datasets where the actual data-generator systems were unknown, and no model of them were available. However, it is important to remark that CFG, DFAs, and training data are artifacts used only with the purpose of evaluating the approach on controlled experiments. Actually, in real application scenarios only the RNN (and its alphabet) is required to be available.

For RNN trained with data generated by CFG and DFA, two experiments were carried out. On one hand, Bounded-L^* was run on the RNN alone to extract

PAC DFA to perform, whenever possible, post-learning verification. Besides, on-the-fly checking through learning is applied on the RNN against several properties. The goal of these experiments is to compare both approaches and validate the theoretical results of the previous section.

4.1 Context-free language modelling

Parenthesis prediction is a typical problem used to study the capacity of RNN for context-free language modelling [10].

First, we trained an RNN with 500K positive and negative sequences upto length 20 generated by a 3-symbol Dyck 1 CFG with alphabet $\{(,),c\}$:

$$S \rightarrow S\ T \mid T\ S \mid T \qquad T \rightarrow (\ T\ ) \mid () \qquad T \rightarrow c$$

The RNN was trained until achieving 100% accuracy on a test set of 50K sequences. This network was checked against 1) its specification, 2) the regular property $(c)^*$, and 3) the context-free language $(^m)^n$ with $m < n$.

Configuration		**Exec. time (s)**			**Average EQ test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**		
0.005	0.005	1.984	7.205	3.072	1,899	0.002792
0.0005	0.005	3.713	10.445	5.997	20,093	0.000264
0.00005	0.005	7.982	30.470	9.997	203,007	0.000026
0.00005	0.0005	8.128	36.621	9.919	249,059	0.000031
0.00005	0.00005	9.625	41.884	12.185	295,111	0.000034

Table 1. Dyck 1: PAC DFA extraction from RNN.

Experimental results are shown in Tables 1 and 2. For each (ϵ, δ), 5 runs were executed. All runs finished with 0-divergence **EQ**. Execution times are in secs. Figures show that on average, on-the-fly checking is usually faster than extracting a PAC DFA from the RNN. It is important to remark that cases 1) and 3) fall in an undecidable playground since checking whether a regular language is contained in a context-free language is undecidable [14]. For case 1), our technique could not find a counterexample, thus giving probabilistic guarantees of emptiness, that is, of the RNN to correctly modelling the 3-symbol parenthesis language. For cases 2) and 3), PAC DFA of the intersection language are found in all runs, showing the properties are indeed not satisfied. Besides, counterexamples are generated orders of magnitude faster (in average) than extracting a DFA from the RNN alone.

Second, we trained an RNN with 500K positive and negative sequences upto length 20 generated from a 5-symbol Dyck 2 CFG with alphabet $\{(,),[,],c\}$:

$$S \rightarrow S\ T \mid T\ S \mid T \qquad T \rightarrow (\ T\ ) \mid () \qquad T \rightarrow [\ T\ ] \mid [] \qquad T \rightarrow c$$

Prop	**Configuration**		**Exec. time (s)**			**First counter-example**	**Average PAC test size**	**Average** ϵ^*
	ϵ	δ	**min**	**max**	**avg**			
1)	0.005	0.005	0.004	0.012	0.006	-	1,476	0.00359
	0.0005	0.005	0.051	0.125	0.067	-	14,756	0.00036
	0.00005	0.005	0.682	0.833	0.747	-	147,556	0.00004
	0.00005	0.0005	1.164	1.595	1.340	-	193,607	0.00004
	0.00005	0.00005	1.272	1.809	1.386	-	239,659	0.00004
2)	0.005	0.005	0.031	34.525	5.762	0.099	1,948	0.00273
	0.0005	0.005	0.397	37.846	10.245	0.084	20,370	0.00026
	0.00005	0.005	4.713	30.714	6.547	0.825	206,473	0.00003
3)	0.005	0.005	0.025	0.966	0.302	0.006	1,899	0.00279
	0.0005	0.005	0.267	1.985	0.787	0.070	20,093	0.00026
	0.00005	0.005	4.376	6.479	4.775	0.764	203,007	0.00003

Table 2. Dyck 1: On-the-fly verification of RNN.

The RNN was trained until achieving 99.646% accuracy on a test set of 50K sequences. This RNN was checked against its specification. For each (ϵ, δ), 5 runs were executed, with a timeout of 300s. Experimental results are shown in Tables 3 and 4. For each configuration, at least three runs of on-the-fly checking finished before the timeout and one was able to find, as expected, the property was not verified by the RNN, exhibiting a counterexample showing it did not model the CFG and yielding a PAC DFA of the wrong classifications.

Configuration		**Exec. time (s)**			**Average EQ test size**	**Average** ϵ^*
ϵ	δ	**min**	**max**	**avg**		
0.005	0.005	2.753	149.214	19.958	1,795	0.003
0.0005	0.005	23.343	300.000	105.367	18,222	0.006
0.00005	0.005	42.518	139.763	77.652	186,372	0.002

Table 3. Dyck 2: PAC DFA extraction from RNN.

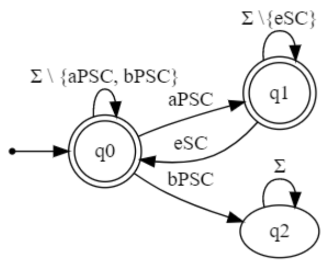
4.2 Software modelling and verification

4.2.1 E-commerce web application API We analyzed an RNN trained with a dataset of 100K positive and negative sequences upto length 16 drawn from the model of the e-commerce system from [26], together with sequences that violate the properties to be checked. These *canary* sequences were added on purpose to check whether the RNN actually learned those faulty behaviors and whether our technique was able to figure that out. The RNN was trained until the measured error on a test set of 16K sequences was 0. We overfitted to ensure the faulty behavior was successfully classified by the RNN.

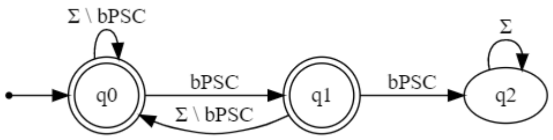
Configuration		**Exec. time (s)**			**First counter-example**	**Average PAC test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**			
0.005	0.005	0.004	122.388	24.483	90.285	1,504	0.00444
0.0005	0.005	55.084	300.000	215.508	42.462	16,604	0.00145
0.00005	0.005	0.695	324.144	158.195	4.545	166,040	0.00003

Table 4. Dyck 2: On-the-fly verification of RNN.

We checked the RNN against the following regular properties: 1) It is not possible to buy products in the shopping cart (modelled by $bPSC$) when the shopping cart is empty. Symbols $aPSC$ and eSC model adding products to and emptying the shopping cart, respectively (Fig. 2a); 2) It is not possible to execute two or more consecutive $bPSC$ (Fig. 2b).



(a) E-commerce property 1



(b) E-commerce property 2

Fig. 2. E-commerce properties

Experimental results are shown in Tables 5 and 6. On-the-fly property checking concluded both properties were not satisfied. In average, it took more time to output a PAC DFA of the language of faulty behaviors than extracting a PAC DFA of the RNN alone. Nevertheless, counterexamples were found (in average) orders of magnitude faster than the latter for requirement 1), while it took comparable time for requirement 2), which revealed to be harder.

Configuration		**Exec. time (s)**			**Average EQ test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**		
0.01	0.01	16.863	62.125	36.071	863	0.00534
0.001	0.01	6.764	9.307	7.864	8,487	0.00054
0.0001	0.01	18.586	41.137	30.556	83,482	0.00006

Table 5. E-commerce: PAC DFA extraction from RNN.

Prop	Configuration		Excecution time (s)			First counter-example	Average PAC test size	Average ϵ^*
	ϵ	δ	min	max	avg			
1	0.01	0.01	87.196	312.080	174.612	3.878	891	0.00517
	0.001	0.01	0.774	203.103	102.742	0.744	9,181	0.00050
	0.0001	0.01	105.705	273.278	190.948	2.627	94,573	0.00005
2	0.01	0.01	0.002	487.709	148.027	80.738	752	0.00619
	0.001	0.01	62.457	600.000	428.400	36.606	8,765	0.00053
	0.0001	0.01	71.542	451.934	250.195	41.798	87,641	0.00005

Table 6. E-commerce: On-the-fly verification of RNN.

4.2.2 Cruise controller We trained an RNN with a dataset containing 200K positive and negative sequences upto a maximum length of 16 from a cruise controller model from [23] (Fig. 3). The measured error of the RNN on a test set of 16K sequences was 0,09%. The property P is shown in Fig. 4. It models the requirement that it is not possible to see a *break* event without having seen *gas* | *acc* before.

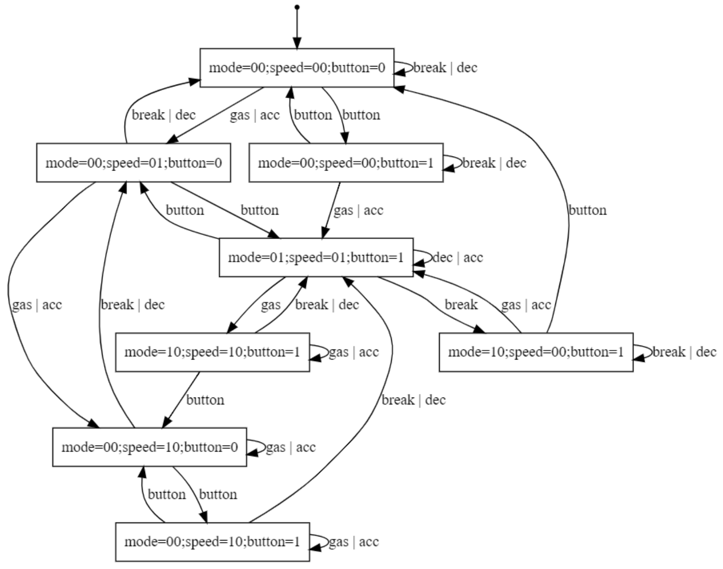


Fig. 3. Model of the cruise controller example

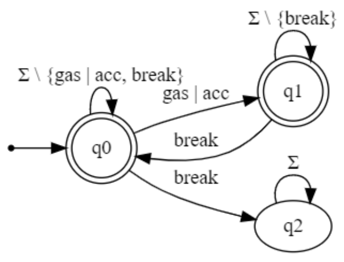


Fig. 4. Requirement of the cruise controller example

Experimental results are shown in Tables 7 and 8. Every run of the on-the-fly technique ended up conjecturing $C \cap \overline{P} = \emptyset$ with perfect **EQ** tests. On the other hand, we extracted PAC DFA from the network alone, with a timeout of 200s. For the first configuration, one run timed out and four completed. All extracted DFA exceeded the maximum number of states bound, and three of them did not verify the property. For the second one, there were two time outs, and three successful extractions. Every one of the extractions exceeded the maximum states bound and two of them did not verify the property. Finally, for the third one, every run timed out. We used oracle **EX** to generate 2 million sequences. For each of the PAC DFA H not checking the property, none was accepted by both $H \cap \overline{P}$ and the RNN. Hence, we cannot disprove the conjecture that the RNN is correct with respect to the requirement P obtained with the on-the-fly algorithm.

Configuration		**Exec. time (s)**			**Average EQ test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**		
0.01	0.01	11.633	200.000	67.662	808	0.05
0.001	0.01	52.362	200.000	135.446	8,071	0.03
0.0001	0.01	-	-	-	-	-

Table 7. Cruise controller: PAC DFA extraction from RNN.

Configuration		**Exec. time (s)**			**First counter-example**	**Average PAC test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**			
0.01	0.01	0.003	0.006	0.004	-	669	0.00688
0.001	0.01	0.061	0.096	0.075	-	6,685	0.00069
0.0001	0.01	0.341	0.626	0.497	-	66,847	0.00007

Table 8. Cruise controller: On-the-fly verification of RNN.

4.3 Domain-specific datasets

4.3.1 Analysis of HDFS logs

In this case study we deal with a dataset of logs of a Hadoop Distributed File System (HDFS) from [7]. The logs were labeled as normal or abnormal by Hadoop experts. It contains 4855 training normal variable-length logs. Each log is pre-processed into a sequence of numeric symbols from 0 to 28. These logs were used to train an RNN-based auto-regressive language model (LM). That is, the output of the RNN is the conditional probability of the next symbol given all the previous ones [5]. This mechanism can be used for sequence classification in several ways. In this case, we used the RNN to compute the probability of a sequence. If the output is greater than a given threshold, the sequence is considered to be normal, otherwise is labeled as abnormal. We used a threshold of 2×10^{-7} which yields an accuracy of 98.35% with no false positives in a perfectly balanced test dataset containing 33600 normal and abnormal logs. That is, no abnormal log is missed by the classifier.

We verified the following properties on the classifier: 1) it does not classify as normal a sequence that contains a symbol that only appeared in abnormal logs (12 of the 29 symbols), and 2) the sum of occurrences in a normal log of symbols "4" and "21", often seen at the beginning of the log, is at most 5.

Prop	**Configuration**		**Excecution time (s)**			**First counter-example**	**Average PAC test size**	**Average ϵ^***
	ϵ	δ	**min**	**max**	**avg**			
1)	0.01	0.01	209.409	1,121.360	555.454	5.623	932	0.0050
	0.001	0.001	221.397	812.764	455.660	1.321	12,037	0.0006
2)	0.01	0.01	35.131	39.762	37.226	-	600	0.0077
	0.001	0.001	252.202	257.312	254.479	-	8,295	0.0008

Table 9. HDFS logs: On-the-fly verification of RNN.

The results of on-the-fly checking are shown in Table 9. For each configuration, 5 runs were executed. In the case of property 1), all runs found counterexamples and output a PAC DFA of the sequences violating the property. This means the classifier can label as normal a log containing symbols that only appeared in logs tagged as abnormal by experts. This observation exhibits a discrepancy with the results on the test dataset where the classifier incurred in no false positives. Moreover, the output PAC DFA can be used to understand in more detail potential mistakes of the classifier to improve its performance. For property 2), we found that it is satisfied by the classifier with PAC guarantees.

4.3.2 TATA-box recognition in DNA promoter sequences

Promoter region recognition in DNA sequences is an active research area in bioinformatics. Recently, tools based on neural networks, such as CNN and LSTM, have been proposed for such matter [27]. Promoters are located upstream near the gene transcription start site (TSS) and control the activation or repression of the genes. The TATA-box is a particular promoter subsequence that indicates to other molecules where transcription begins. It is a T/A-rich (i.e., more T's and A's than C's and G') subsequence of 6 base pairs (bp) located between positions –30bp to –25bp, where +1bp is the TSS.

The goal of this experiment is not to develop a neural network for promoter classification, but to study whether an RNN trained with TATA and non-TATA promoter sequences is able to distinguish between them, that is, it is capable of determining whether a DNA sequence contains a TATA region. For such task, we trained an RNN composed of an LSTM and a dense layer for classification, with a dataset of the most representative TATA (2067 sequences) and non-TATA (14388 sequences) human promoters of length 50bp from positions -48bp to +1bp. The dataset was downloaded from the website EPDnew[5], The RNN was trained until achieving an accuracy of 100%.

We ran the on-the-fly algorithm to check whether the language of the RNN was included in the set of sequences containing a TATA-box. The property was coded as a Python program which counts the number of T's, A's, C's and G's in the subsequence from position -30bp to -25bp of the genomic sequence, and checks whether the sum of T's and A's is greater than the sum of C's and G's. In this case, the oracle **EX** was parameterized to generate sequences of length 50 over the alphabet $\{A, T, C, G\}$. Table 10 shows the experimental results. Hence, the RNN is a probably correct approximation of the property.

Configuration		**Exec. time (s)**			**First counter-example**	**Average PAC test size**	**Average ϵ^***
ϵ	δ	**min**	**max**	**avg**			
0.01	0.01	5.098	5.259	5.168	-	600	0.00768
0.001	0.001	65.366	66.479	65.812	-	8,295	0.00083
0.0001	0.0001	865.014	870.663	867.830	-	105,967	0.00008

Table 10. TATA-box: On-the-fly verification of RNN.

5 Related Work

Learning and automata-theoretic verification have been combined in several ways for checking temporal requirements of systems. For instance, [2, 6, 8] do compositional verification by learning assumptions. These methods are white-box and require an external decision procedure. Learning regular approximations of non-regular languages (FIFO automata) for verifying safety properties have been explored in [38]. Still, it relies on a state-based representation and requires being able to compute successor states of words by transitions of the target automata, which is not feasible for RNN. The post-learning verification technique presented in [9] iteratively applies Trakhtenbrot-Barzdin algorithm [36] on several training sets until the inferred automaton is an invariant sufficient to prove the property.

Learning based testing (LBT) [24] is a black-box checking approach for generating test cases. It relies on incrementally building hypotheses of the system

[5] `https://epd.epfl.ch//index.php`

under test and verifying whether they satisfy the requirement through an external model-checker. Counterexamples produced by the model-checker serve as test-cases for the system under test. To the best of our knowledge, it does not provide provable probabilistic guarantees. Recent work [22] proposes a sound extension but it requires relaxing the black-box setting by observing and recording the internal state of the system under test.

Concerning verification of neural networks, several approaches have been proposed to check a specific kind of requirement, namely robustness, which evaluates artificial neural network resilience to adversarial examples. A technique for black-box robustness testing is developed in [44]. DeepSafe is a white-box tool for checking robustness based on clustering and constraint solvers. A white-box algorithm for feed-forward multi-layer neural networks based on satisfiability modulo theories is presented in [15]. The method exhaustively searches for adversarial misclassifications, propagating the analysis from one layer to the other directly through the network source code. These works have been applied for image classification with deep convolutional and dense layers, but not for recurrent neural networks over symbolic sequences.

For RNN, a post-learning approach for adversarial accuracy verification is presented in [41] based a white-box rule-extraction technique to extract DFA from RNN. Experimental evaluation is carried out work on Tomita grammars [35], which are all regular languages over the $\{0, 1\}$-alphabet. That approach does not offer any guarantee on how well the DFA approximates the RNN. In [16] white-box RNN verification is done by generating a series of abstractions. Specifically, the method strongly relies on the internal structure and weights of the RNN to generate a feed-forward network (FFNN), which is proven to compute the same output. Then, reachability analysis is performed resorting to Linear Programming (LP) and Satisfiability Modulo Theories (SMT) techniques.

Finally, statistical model checking (SMC) the system under analysis and/or the property is stochastic [1, 20]. The objective of SMC is to check whether a stochastic system, such as a Markov decision process, satisfies a property with a probability greater or equal to a certain threshold θ. The problem we address in this work is different as neither the system nor the property is stochastic. Our approach provides statistical guarantees that the language of an RNN C is included in another language (the property P) or provides a PAC model of the language $C \cap \overline{P}$, along with actual counterexamples showing it is not.

6 Conclusions

We presented a learning-based approach for checking properties on RNN. The approach is black-box since it is not restricted to any particular class of RNN or property. It is also on-the-fly because it does not build a-priori the state-space of the RNN but rather it constructs an approximation of the intersection of the RNN with the negation of the requirement.

Our approach provides better guarantees than post-learning verification as whenever the language learnt is non-empty it is certain that the property is not satisfied, and real counterexamples are provided. Moreover, if the learning algorithm is run to completion, it outputs a probably correct approximation of the set of incorrect behaviors.

We implemented the approach and applied it to verifying properties on several case studies. We compared, when possible, the results of on-the-fly checking through learning with post-learning model-checking on extracted PAC models of the RNN alone. The experiments were promising as they provided empirical evidence that the on-the-fly approach typically performs much faster than post-learning verification when the property is found to be probably approximately correct.

Acknowledgments This work has been partially funded by an ICT4V - Information and Communication Technologies for Verticals master thesis grant under the code `POS_ICT4V_2016_1_15`.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1) (Jan 2018). https://doi.org/10.1145/3158668, `https://doi.org/10.1145/3158668`
2. Alur, R., Madhusudan, P., Nam, W.: Symbolic compositional verification by learning assumptions. In: International Conference on Computer Aided Verification. pp. 548–562. Springer (2005)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (Nov 1987)
4. Angluin, D.: Computational learning theory: Survey and selected bibliography. In: Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing. pp. 351–369. STOC '92, ACM, New York, NY, USA (1992)
5. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**(null), 1137–1155 (Mar 2003)
6. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 331–346. Springer (2003)
7. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1285–1298. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134015, `https://doi.org/10.1145/3133956.3134015`
8. Feng, L., Han, T., Kwiatkowska, M., Parker, D.: Learning-based compositional verification for synchronous probabilistic systems. In: International Symposium on Automated Technology for Verification and Analysis. pp. 511–521. Springer (2011)
9. Habermehl, P., Vojnar, T.: Regular model checking using inference of regular languages. Electronic Notes in Theoretical Computer Science **138**, 21–36 (09 2004). https://doi.org/10.1016/j.entcs.2005.01.044
10. Hao, Y., Merrill, W., Angluin, D., Frank, R., Amsel, N., Benz, A., Mendelsohn, S.: Context-free transductions with neural stacks. preprint arXiv:1809.02836 (2018)

11. Heinz, J., de la Higuera, C., van Zaanen, M.: Formal and empirical grammatical inference. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts of ACL 2011. pp. 2:1–2:83. HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011)
12. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997)
14. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Acm Sigact News **32**(1), 60–65 (2001)
15. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Int. Conf. on Computer Aided Verification. pp. 3–29. Springer (2017)
16. Kevorchian, A.: Verification of Recurrent Neural Networks. Master's thesis, Imperial College London (2018)
17. Kim, J., Kim, J., Thu, H.L.T., Kim, H.: Long short term memory recurrent neural network classifier for intrusion detection. In: 2016 International Conference on Platform Technology and Service (PlatCon). pp. 1–5. IEEE (2016)
18. Kocić, J., Jovičić, N., Drndarević, V.: An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms. Sensors **19**(9), 2064 (2019)
19. Kwiatkowska, M.Z.: Safety verification for deep neural networks with provable guarantees. In: 30th International Conference on Concurrency Theory (CONCUR 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
20. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical Model Checking, pp. 478–504. Springer, Cham (2019)
21. Mayr, F., Yovine, S.: Regular inference on artificial neuralnetworks. In: Holzinger, A., et al. (eds.) Machine Learning and Knowledge Extraction. pp. 350–369. Springer International Publishing, Cham (2018)
22. Meijer, J., van de Pol, J.: Sound black-box checking in the learnlib. Innovations in Systems and Software Engineering **15**(3-4), 267–287 (2019)
23. Meinke, K., Sindhu, M.A.: Lbtest: A learning-based testing tool for reactive systems. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. pp. 447–454 (March 2013). https://doi.org/10.1109/ICST.2013.62
24. Meinke, K.: Learning-based testing: recent progress and future prospects. In: Machine Learning for Dynamic Software Analysis: Potentials and Limits, pp. 53–73. Springer (2018)
25. Merrill, W.: Sequential neural networks as automata. arXiv preprint arXiv:1906.01615 (2019)
26. Merten, M.: Active automata learning for real life applications. Ph.D. thesis, Technischen Universität Dortmund (2013)
27. Oubounyt, M., Louadi, Z., Tayara, H., Chong, K.T.: Deepromoter: Robust promoter predictor using deep learning. Frontiers in genetics **10** (2019)
28. Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A.: Malware classification with recurrent networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia. pp. 1916–1920 (April 19-24, 2015)
29. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. Journal of Automata, Languages and Combinatorics **7**(2), 225–246 (2002)

30. Rhode, M., Burnap, P., Jones, K.: Early stage malware prediction using recurrent neural networks. Computers & Security **77** (08 2017). https://doi.org/10.1016/j.cose.2018.05.010
31. Scheiner, N., Appenrodt, N., Dickmann, J., Sick, B.: Radar-based road user classification and novelty detection with recurrent neural network ensembles. In: 2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019. pp. 722–729 (2019). https://doi.org/10.1109/IVS.2019.8813773, `https://doi.org/10.1109/IVS.2019.8813773`
32. Siegelmann, H.T., Sontag, E.D.: On the computational power of neural nets. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. pp. 440–449. COLT '92, ACM, New York, NY, USA (1992)
33. Singh, D., Merdivan, E., Psychoula, I., Kropf, J., Hanke, S., Geist, M., Holzinger, A.: Human activity recognition using recurrent neural networks. In: International Cross-Domain Conference for Machine Learning and Knowledge Extraction. pp. 267–274. Springer (2017)
34. Suzgun, M., Belinkov, Y., Shieber, S.M.: On evaluating the generalization of lstm models in formal languages. CoRR **abs/1811.01001** (2018)
35. Tomita, M.: Dynamic construction of finite automata from examples using hill-climbing. In: Proceedings of the Fourth Annual Conference of the Cognitive Science Society. pp. 105–108. Ann Arbor, Michigan (1982)
36. Trakhtenbrot, B.A., Barzdin, I.M.: Finite automata : behavior and synthesis [by] B. A. Trakhtenbrot and Ya. M. Barzdin. Translated from the Russian by D. Louvish. English translation edited by E. Shamir and L. H. Landweber. North-Holland Pub. Co.; American Elsevier Amsterdam, New York (1973)
37. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (Nov 1984)
38. Vardhan, A., Sen, K., Viswanathan, M., Agha, G.: Actively learning to verify safety for fifo automata. In: International Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 494–505. Springer (2004)
39. Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Venkatraman, S.: Robust intelligent malware detection using deep learning. IEEE Access **7**, 46717–46738 (2019)
40. Wang, Q., Zhang, K., II, A.G.O., Xing, X., Liu, X., Giles, C.L.: A comparison of rule extraction for different recurrent neural network models and grammatical complexity. CoRR **abs/1801.05420** (2018), `http://arxiv.org/abs/1801.05420`
41. Wang, Q., Zhang, K., Liu, X., Giles, C.L.: Verification of recurrent neural networks through rule extraction. In: AAAI Spring Symposium on Verification of Neural Networks (VNN19) (2019)
42. Wang, Q., Zhang, K., Ororbia, II, A.G., Xing, X., Liu, X., Giles, C.L.: An empirical evaluation of rule extraction from recurrent neural networks. Neural Comput. **30**(9), 2568–2591 (Sep 2018)
43. Weiss, G., Goldberg, Y., Yahav, E.: Extracting automata from recurrent neural networks using queries and counterexamples. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80. PMLR, Stockholmsmässan, Stockholm, Sweden (10–15 Jul 2018)
44. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 408–426. Springer (2018)
45. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. Ieee Access **5**, 21954–21961 (2017)