Regular Inference on Artificial Neural Networks

Franz Mayr, Sergio Yovine

Regular Inference on Artificial Neural Networks

Franz Mayr[1] and Sergio Yovine[2]

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

Abstract. This paper explores the general problem of explaining the behavior of artificial neural networks (ANN). The goal is to construct a representation which enhances human understanding of an ANN as a sequence classifier, with the purpose of providing insight on the rationale behind the classification of a sequence as positive or negative, but also to enable performing further analyses, such as automata-theoretic formal verification. In particular, a probabilistic algorithm for constructing a deterministic finite automaton which is approximately correct with respect to an artificial neural network is proposed.

Keywords: Artificial neural networks · Sequence classification · Deterministic finite automata · Probably approximately correct learning

1 Introduction

The purpose of explainable artificial intelligence is to come up with artifacts capable of producing intelligent outcomes together with appropriate rationalizations of them. It means that besides delivering best possible model performance metrics (e.g., accuracy) and computational performance metrics (e.g., algorithmic complexity), they must provide adequate and convincing reasons for effectively justifying the judgment in a human-understandable way.

Artificial neural networks (ANN) are the state-of-the-art method for many fields in the area of artificial intelligence [20]. However, ANN are considered to be a rather obscure model [21], meaning that understanding the specifics that were taken into consideration by the model to make a decision is not a trivial task. Human understanding of the model is crucial in fields such as medicine [18], risk assessment [4], or intrusion detection [33]. From the point of view of explaining the rationale of an outcome, an important issue is that ANN lack an explicit and constructive characterization of their embedded decision-making strategy.

This limitation of ANN explanatory capabilities motivated a large amount of research work aiming at improving ANN explainability. Several approaches to tackle this issue have been identified [10, 14]. In particular, [14] characterizes the *black-box model* explanation problem. It consists in providing a human-understandable model which is able to mimic the behavior of the ANN. In [10], the problem of *processing* explanation is defined. This approach seeks answering *why* a given input leads the ANN to produce a particular outcome. Another

approach consists in allowing a human actor to interact with the learning process. This human-in-the-loop method of addressing explainability, called *glass box* interactive machine-learning approach, is presented in [19].

In this paper we follow a black-box model and processing explanation approach for ANN. We are interested in studying explainability in the context of ANN trained to solve *sequence* classification problems [32, 34], which appear in many application domains. In the past few years several classes of ANN, such as Recurrent Neural Networks (RNN), e.g., Long-Short Term Memory (LSTM) [17], have been successfully applied for such matter [6, 8, 22, 27, 28, 30, 31, 35].

We restrict the study to binary classification. This problem is a case of *language membership*, where the language of the ANN is the set of sequences classified as positive by the network. The trained ANN hides a model of such sequences which it uses to predict whether a given input sequence belongs to the language. If the language from which the training samples have been drawn is known, the question is how well the ANN learned it [8,9,26]. Another, may be more realistic situation occurs when the target language is unknown. In this case, the question to answer becomes what is the language learned by the ANN, or more precisely, whether it could be characterized operationally instead of denotationally.

Typically, these questions are addressed by looking at the accuracy of the network on a given test set. However, it has been observed that networks trained with millions of samples which exhibit 100% accuracy on very large development test sets could still incorrectly classify random sequences [12, 31]. Thus, exact convergence on a set of sequences, whatever its size, does not ensure the language of the network is the same as the target language. Hence, in both cases, the question remains whether the language recognized by the network could be explained, even approximately, in some other, comprehensible, way.

The goal of this paper is to provide means for extracting a constructive representation of the model hidden inside the ANN. To the best of our knowledge, all previous works devoted to model and processing explanation for ANN are either white-box, that is, they look into and/or make assumptions about the network's structure and state, or they are focused on extracting decision trees or rules. The reader is referred to [3, 10, 14] for recent surveys on this topic.

Now, when it comes to operationally explain the dynamical system that produces sequences of events, rules and trees are not expressive enough. In this case, a widely used formalism are automata [13]. This model provides a language-independent mathematical support for studying dynamical systems whose behavior could be understood as sequences corresponding to words of a regular language. In the automata-theoretic approach, when the system under analysis is a black box, that is, its internal structure (composed of states and transitions) is unknown, the general problem of constructing an automaton that behaves as the black box is called *identification* or *regular inference* [16].

Many times it is not theoretically or practically feasible to solve this problem precisely, in which case, it needs to be solved approximately. That is, rather than exactly identifying the automaton inside the black box, we attempt to find an automaton which is a reasonable approximation with some confidence.

The Probably Approximately Correct (PAC) framework [29] is a general approach to solve problems like the one we are considering here. A *learner*, which attempts to identify the hidden machine inside the black box, can interact with a *teacher*, which has the ability to answer queries about the unknown machine to be learned. For this, the teacher uses an oracle which draws positive and negative samples with some probability distribution. There are several specific problem instances and algorithms to solve them, depending on the assumptions that are made regarding how the behavior of the black box is observed, what questions could be asked, how the answers to these questions could be used to build an automaton, etc. The reader is referred to [2, 16] for a thorough review.

In this context, two general settings can be distinguished, namely *passive* or *active* learning. The former consists in learning a language from a set of given (chosen by the teacher) positive and/or negative examples [25]. It has been shown in [11] that this problem is *NP-complete*. In the latter, the learner is given the ability to draw examples and to ask membership queries to the teacher. A well known algorithm in this category is Angluin's L^* [1]. L^* is *polynomial* on the number of states of the minimal deterministic finite automaton (DFA) and the maximum length of any sequence exhibited by the teacher.

The relationship between automata and ANN has been thoroughly studied: [26] presents mechanisms for programming an ANN that can correctly classify strings of arbitrary length belonging to a given regular language; [9] discuss an algorithm for extracting the finite state automaton of second-order recurrent neural networks; [31] look at this problem in a white-box setting. Therefore, according to [14], a black-box model explanation approach, that is, using regular inference algorithms that do not rely on the ANN structure and weights is a problem that has not been addressed so far.

Of course, one may argue that an automaton could be directly learned from the dataset used to train the network. However, this approach has several drawbacks. First, passive learning is NP-complete [11]. Second, it has been shown that ANN such as LSTM, are much better learners, as they learn faster and generalize better. Besides, they are able to learn languages beyond regular ones. Third, the training dataset may not be available, in which case the only way to construct an explanation is to query the ANN.

The contribution of this work is an adaptation of Angluin's L^* algorithm that outputs a DFA which approximately behaves like an input ANN whose actual structure is completely unknown. This means that whenever a sequence is recognized by the DFA constructed by the algorithm, it will most likely be classified as positive by the ANN, and vice versa. We stress the fact that our algorithm is completely agnostic of the structure of the ANN.

Outline In Sec. 2 we precisely present the problem we are going to address and the method used for solving it. In Sec. 3 we discuss the proposed algorithm and a variation of the general PAC framework to analyze its behavior. In Sec. 4 we present the experimental results carried out on several examples which validate the theoretical analyses. In Sec. 5 we compare and contrast our approach with related works. Finally we present the conclusions and future work.

2 Preliminaries

2.1 Problem statement

Let of $\mathcal{U} \subseteq \Sigma^*$ be some *unknown* language over an alphabet Σ of symbols, and $\mathcal{S} \subseteq \Sigma^*$ be a sample set such that it contains positive and negative sequences of $\mathcal{U}$. That is, there are sequences in $\mathcal{S}$ which belong to $\mathcal{U}$ and others that do not.

The *language* of an ANN $\mathcal{N}$, denoted $\mathcal{L}(\mathcal{N})$, is the set of sequences classified as positive by $\mathcal{N}$. Suppose $\mathcal{N}$ is obtained by training it with a sample set $\mathcal{S}$, and then used to predict whether a sequence $u \in \Sigma^*$ does belong to $\mathcal{U}$. In other words, the unknown language $\mathcal{U}$ is considered to be somehow approximated by $\mathcal{L}(\mathcal{N})$, that is, with high probability $x \in \mathcal{L}(N) \iff x \in \mathcal{U}$.

But, what is the actual language $\mathcal{L}(\mathcal{N})$ learned by $\mathcal{N}$? Is it a regular language? That is, could it be expressed by a deterministic finite automaton? Is it possible to approximate it somehow with a regular language? The interest of having an automaton-based, either precise or approximated, characterization of $\mathcal{L}(\mathcal{N})$, allows to explain the answers of $\mathcal{N}$, while providing insight on the unknown language $\mathcal{U}$. This approach enhances human understanding because of the visual representation but also because it enables performing further analyses, such as automata-theoretic formal verification [5].

2.2 Probably Approximately Correct (PAC) learning

In order to study the questions above, we resort to Valiant's PAC-learning framework [2, 29]. Since we are interested in learning languages, we restrict ourselves to briefly describing the PAC-learning setting for languages.

Let $\mathcal{D}$ be an unknown distribution over Σ^* and $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$. The *symmetric difference* between $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_1 \oplus \mathcal{L}_2$, is the set of sequences that belong to only one of the languages, that is, $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathcal{L}_1 \setminus \mathcal{L}_2 \cup \mathcal{L}_2 \setminus \mathcal{L}_1$.

The *prediction error* of $\mathcal{L}_1$ with respect to $\mathcal{L}_2$ is the probability of a sequence to belong to their symmetric difference, denoted $\mathbf{P}_{\mathcal{D}}(\mathcal{L}_1 \oplus \mathcal{L}_2)$. Given $\epsilon \in (0, 1)$, we say that $\mathcal{L}_1$ is ϵ-*approximately correct* with respect to $\mathcal{L}_2$ if $\mathbf{P}_{\mathcal{D}}(\mathcal{L}_1 \oplus \mathcal{L}_2) < \epsilon$.

The *oracle* $\mathbf{EX}_{\mathcal{D}}(\mathcal{L}_1)$ draws an *example* sequence $x \in \Sigma^*$ following distribution $\mathcal{D}$, and tags it as *positive* or *negative* according to whether it belongs to $\mathcal{L}_1$ 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 $\delta \in (0, 1)$, *target* language $\mathcal{L}_t$ and oracle $\mathbf{EX}_{\mathcal{D}}(\mathcal{L}_t)$, and if it terminates, it outputs a language $\mathcal{L}_o$ such that $\mathcal{L}_o$ is ϵ-approximately correct with respect to $\mathcal{L}_t$ with probability at least $1 - \delta$.

A PAC-learning algorithm can also be equipped with an *approximate equivalence* test **EQ** which checks a candidate output $\mathcal{L}_o$ against the target language $\mathcal{L}_t$ using a *sufficiently large* sample of tagged sequences S generated by **EX**. If the sample is such that for every $x \in S$, $x \in \mathcal{L}_t \iff x \in \mathcal{L}_o$, the algorithm successfully stops and outputs $\mathcal{L}_o$. Otherwise, it picks any sequence in $S \cap (\mathcal{L}_o \oplus \mathcal{L}_t)$ as *counterexample* and continues.

The algorithm may also be allowed to call directly a *membership* oracle **MQ**, such that $\mathbf{MQ}(x, \mathcal{L}_t)$ is true if and only if $x \in \mathcal{L}_t$. Notice that the **EX** oracle may also call **MQ** to tag sequences.

A *distribution-free* algorithm is one that works for every $\mathcal{D}$. Hereinafter, we will focus on distribution-free algorithms, so we will omit $\mathcal{D}$.

2.3 L^*

L^* [1] learns *regular languages*, or equivalently, *deterministic finite automata* (DFA). Given a DFA $\mathcal{A}$, we use $\mathcal{L}(\mathcal{A})$ to denote its language, that is, the set of sequences accepted by $\mathcal{A}$. We denote $\mathcal{A}_t$ and $\mathcal{A}_o$ the target and output automata, respectively. The symmetric difference between A_o and A_t, denoted $A_o \oplus \mathcal{A}_t$, is defined as $\mathcal{L}_o \oplus \mathcal{L}_t$. We say that $\mathcal{A}_o$ ϵ-approximates A_t if $\mathcal{L}_o$ ϵ-approximates $\mathcal{L}_t$.

L^* uses **EQ** and **MQ**. Each time **EQ** is called, it must draw a sample of a size large enough to ensure a *total* confidence of the algorithm of at least $1 - \delta$. That is, whenever the statistical test is passed, it is possible to conclude that the candidate output is ϵ-approximately correct with confidence at least $1 - \delta$.

Say **EQ** is called at iteration i. In order to guarantee the aforementioned property, a sample S_i of size r_i is drawn, where:

$$r_i = \left\lceil \frac{1}{\epsilon} \left(i \ln 2 - \ln \delta \right) \right\rceil \tag{1}$$

This ensures that the probability of the output automaton $\mathcal{A}_o$ *not* being ϵ-approximately correct with respect to $\mathcal{A}_t$ when *all* sequences in a sample pass the **EQ** test, i.e., $S_i \cap (\mathcal{A}_o \oplus \mathcal{A}_t) = \emptyset$, is *at most* δ, that is:

$$\sum_{i>0} \mathbf{P}(S_i \cap (\mathcal{A}_o \oplus \mathcal{A}_t) = \emptyset \mid \mathbf{P}(\mathcal{A}_o \oplus \mathcal{A}_t) > \epsilon) < \sum_{i>0} (1-\epsilon)^{r_i} < \sum_{i>0} 2^{-i}\delta < \delta$$

Remark It is worth noticing that from the point of view of statistical hypothesis testing, a sample S_i that passes the test, gives a confidence of at least $1 - 2^{-i}\delta$.

3 PAC-learning for ANN

We address the following problem: given a ANN $\mathcal{N}$, is it possible to build a DFA $\mathcal{A}$, such that $\mathcal{L}(\mathcal{A})$ is ϵ-approximately correct with respect to $\mathcal{L}(N)$?

3.1 Basic idea

The basic idea to solve this problem is to use L^* as follows. The **MQ** oracle consists in querying $\mathcal{N}$ itself. The **EQ** oracle consists in drawing a sample set S_i with size r_i as defined in equation (1) and checking whether $\mathcal{N}$ and the candidate automaton $\mathcal{A}_i$ completely agree in S_i, that is, $S_i \cap (\mathcal{A}_i \oplus \mathcal{N})$ is empty.

The results reviewed in the previous section entail that if L^* terminates, it will output a DFA $\mathcal{A}$ which is an ϵ-approximation of $\mathcal{N}$ with probability at least

$1-\delta$. Moreover, L^* is proven to terminate *provided $\mathcal{L}(\mathcal{N})$ is a regular language*. However, since ANN are strictly more expressive than DFA [23], there is no guarantee that L^* will eventually terminate; it may not exist a DFA $\mathcal{A}$ with the same language as $\mathcal{N}$. In other words, there is no upper bound $n_{\mathcal{N}}$ such that L^* will terminate in at most $n_{\mathcal{N}}$ iterations for every target ANN $\mathcal{N}$. Therefore, it may happen that for every i the call to **EQ** fails, that is, $S_i \cap (\mathcal{A}_i \oplus \mathcal{N}) \neq \emptyset$.

3.2 Bounded-L^*

To cope with this situation, we resort to imposing a bound to the number of iterations of L^*. Obviously, a direct way of doing it would be to just fix an arbitrary upper bound to the number of iterations. Instead, we propose to constrain the maximum number of states of the automaton to be learned and to restrict the length of the sequences used to call **MQ**. The latter is usually called the *query length*. Typically, these two measures are used to determine the complexity of a PAC-learning algorithm [15].

3.2.1 Algorithm Similarly to the description presented in [16] the algorithm Bounded-L^*(Algorithm 1) can be described as follows:

Algorithm 1: Bounded-L^*

```
Input  : MaxQueryLength, MaxStates, ϵ, δ
Output: DFA A
Lstar-Initialise;
repeat
    while OT is not closed or not consistent do
        if OT is not closed then
            OT, QueryLengthExceeded ← Lstar-Close(OT);
        end
        if OT is not consistent then
            OT, QueryLengthExceeded ← Lstar-Consistent(OT);
        end
    end
    if not QueryLengthExceeded then
        LastProposedAutomaton ← Lstar-BuildAutomaton(OT);
        Answer ← EQ(LastProposedAutomaton);
        MaxStatesExceeded ←
          STATES(LastProposedAutomaton)>MaxStates;
        if Answer ≠ Yes and not MaxStatesExceeded then
            OT ← Lstar-UseEQ(OT, Answer);
        end
    end
    BoundReached ← QueryLengthExceeded or MaxStatesExceeded;
until Answer = Yes or BoundReached;
return LastProposedAutomaton;
```

The observation table OT is initialised by Lstar-Initialise in the same manner that it is for L^*. This step consists in building the structure of the observation table OT as proposed by Angluin. Then the construction of hypotheses begins.

If OT is not *closed* an extra row is added by the Lstar-Close procedure. If OT is *inconsistent*, an extra column is added by the Lstar-Consistent procedure. Both procedures call **MQ** to fill the holes in the observation table. The length of these queries may exceed the maximum query length, in which case the QueryLengthExceeded flag is set to true.

When the table is closed and consistent, and in the case that the query length was not exceeded, an equivalence query **EQ** is made and the automaton number of states is compared to the maximum number of states bound. If **EQ** is unsuccessful and the maximum number of states was not reached, new rows are added by Lstar-UseEQ, using the counterexample contained in *Answer*.

Finally, if the hypothesis passes the test or one of the bounds was reached, the algorithm stops and returns the last proposed automaton.

3.2.2 Analysis Bounded-L^* will either terminate with a successful **EQ** or when a bound (either the maximum number of states or query length) is exceeded. In the former case, the output automaton $\mathcal{A}$ is proven to be an ϵ-approximation of $\mathcal{N}$ with probability at least $1-\delta$ by Angluin's results. In the latter case, the output $\mathcal{A}$ of the algorithm will be the *last automaton proposed by the learner*. $\mathcal{A}$ may not be an ϵ-approximation with confidence at least $1-\delta$, because $\mathcal{A}$ failed to pass the last statistical equivalence test **EQ**. However, the result of such test carries statistical value about the relationship between $\mathcal{A}$ and $\mathcal{N}$. The question is, what could indeed be said about this automaton?

Assume at iteration i **EQ** fails and the number of states of $\mathcal{A}_i$ is greater than or equal to the maximum number of states, or at the next iteration $i+1$, **MQ** fails because the maximum query length is exceeded. This means that $\mathcal{A}_i$ and $\mathcal{N}$ disagree in, say, $k > 0$, of the r_i sequences of S_i. In other words, there are k sequences in S_i which indeed belong to the symmetric difference $\mathcal{A}_i \oplus \mathcal{N}$.

Confidence parameter. Let $p \in (0,1)$ be the actual probability of a sequence to be in $\mathcal{A}_i \oplus \mathcal{N}$ and K_{r_i} the random variable defined as the number of sequences in $\mathcal{A}_i \oplus \mathcal{N}$ in a sample of size r_i. Then, the probability of $K_{r_i} = k$ is:

$$\mathbf{P}(K_{r_i} = k) = \binom{r_i}{k}(1-p)^{r_i-k}p^k$$

Let us first set as our hypothesis that $\mathcal{A}_i$ is an ϵ-approximation of $\mathcal{N}$. Can we accept this hypothesis when $K_{r_i} = k$ with confidence at least $1-\delta'$, for some $\delta' \in (0,1)$? In other words, is there a δ' such that the probability of $K_{r_i} = k$ is smaller than δ' when $p > \epsilon$? Suppose $p > \epsilon$. Then, it follows that:

$$\mathbf{P}(K_{r_i} = k \mid p > \epsilon) = \binom{r_i}{k}(1-p)^{r_i-k}p^k < \binom{r_i}{k}(1-\epsilon)^{r_i-k} < \binom{r_i}{k}e^{-\epsilon(r_i-k)}$$

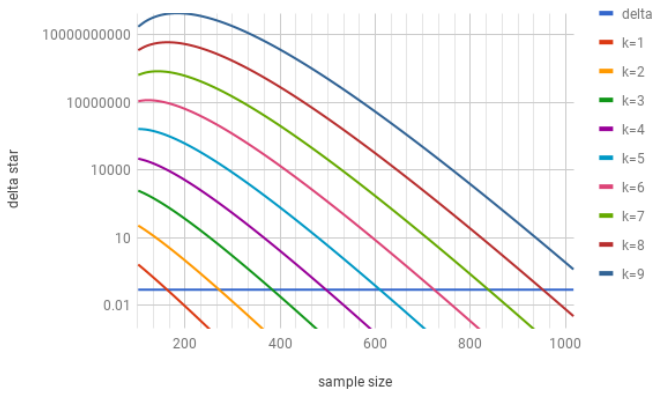


Fig. 1. Values of δ_i^* in log scale as function of r_i, $k \in [1, 9]$, $i \in [3, 70)$

Therefore, if the following condition holds

$$\binom{r_i}{k} e^{-\epsilon(r_i - k)} < \delta' \tag{2}$$

we have that $\mathbf{P}(K_{r_i} = k, p > \epsilon) < \delta'$. That is, the probability of incorrectly accepting the hypothesis with k discrepancies in sample S_i of size r_i is smaller than δ'. Then, we could accept the hypothesis with a confidence of at least $1 - \delta'$.

The left-hand-side term in condition (2) gives us a lower bound δ_i^* for the confidence parameter such that we could accept the hypothesis with probability at least $1 - \delta'$, for every $\delta' > \delta_i^*$:

$$\delta_i^* = \binom{r_i}{k} e^{-\epsilon(r_i - k)} \tag{3}$$

A major problem, however, is that δ_i^* may be greater than 1, and so no $\delta' \in (0, 1)$ exists, or it may be just too large, compared to the desired δ, to provide an acceptable level of confidence for the test.

Fig. 1 shows δ^*, in log scale, for $\epsilon = 0.05$, $k \in [1, 9]$ and r_i computed using equation (1), and compares it with a desired $\delta = 0.05$. We see that as k increases, larger values of r_i, or equivalently, more iterations, are needed to get a value of δ_i^* smaller than δ (horizontal line). Actually, for fixed k and $\epsilon \in (0, 1)$, δ_i^* tends to 0 as r_i tends to ∞. In other words, there is a large enough sample size for which it is possible to make δ_i^* smaller than any desired confidence parameter δ.

Approximation parameter. An alternative would be to look at the approximation parameter ϵ, rather than the confidence parameter δ. In this case, we set as our hypothesis that $\mathcal{A}_i$ is an ϵ'-approximation of $\mathcal{N}$, for some $\epsilon' \in (0, 1)$. Is the probability of accepting this hypothesis with the test tolerating k discrepancies,

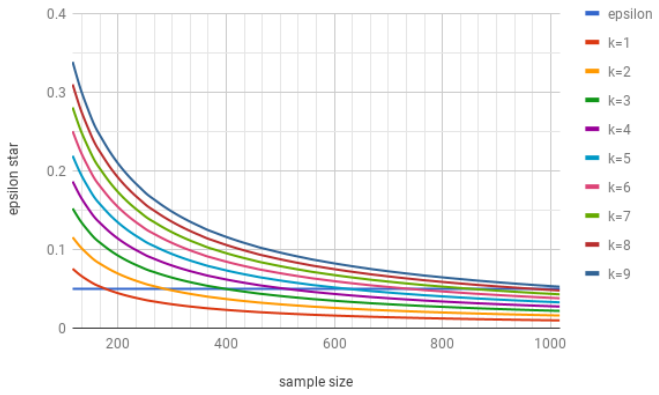


Fig. 2. Values of ϵ_i^* as function of r_i, $k \in [1, 9]$, $i \in [3, 70)$

when the hypothesis is actually false, smaller than δ? That is, we are asking whether the following condition holds for ϵ':

$$\mathbf{P}(K_{r_i} = k \mid p > \epsilon') < \binom{r_i}{k} e^{-\epsilon'(r_i - k)} < \delta$$

Now, we could determine a lower bound ϵ_i^*, such that this condition holds for every $\epsilon' > \epsilon_i^*$, in which case we could conclude that $\mathcal{A}_i$ is an ϵ'-approximation of $\mathcal{N}$, with confidence at least $1 - \delta$. Provided $r_i - k \neq 0$, we have:

$$\epsilon_i^* = \frac{1}{r_i - k}\left(\ln \binom{r_i}{k} - \ln \delta\right)$$

Fig. 2 shows ϵ_i^* for $\delta = 0.05$, $k \in [1, 9]$ and r_i computed using equation (1), and compares it with a desired $\epsilon = 0.05$. We see that as k increases, larger samples, i.e., more iterations, are needed to get a value smaller than ϵ (horizontal line). Nevertheless, for fixed k and $\delta \in (0, 1)$, ϵ_i^* tends to 0 as r_i tends to ∞. In other words, there is a large enough sample size for which it is possible to make ϵ^* smaller than any desired approximation parameter ϵ.

Number of discrepancies and sample size. Actually, we could also search for the largest number k^* of discrepancies which the **EQ** test could cope with for given ϵ, δ and whichever sample size r, or the smallest sample size r^* for fixed ϵ, δ and number of discrepancies k, independently of the number i of iterations.

The values k^* and r^* could be obtained by solving the following equation for k and r, respectively:

$$\ln \binom{r}{k} - \epsilon \ln(r - k) - \ln \delta = 0 \tag{4}$$

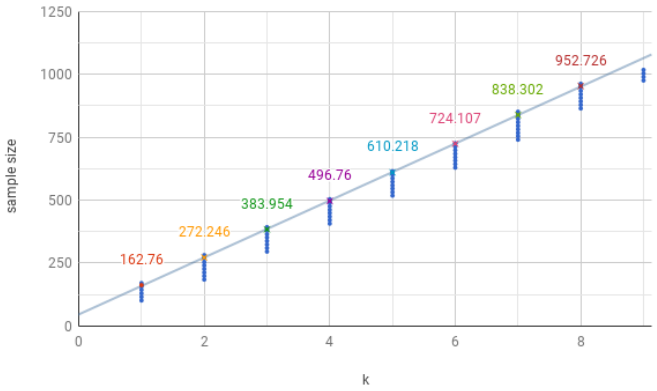


Fig. 3. Comparison between r_i, r^* and k

Fig. 3 plots the relationship between $k \in [1, 9]$, r_i computed using equation (1), and $r^*(k)$, where $r^*(k)$ denotes the value of r^* for the given k. Each point in the vertical dotted segments corresponds to a value of sample size r_i. For a given value of k, we plot all values of r_i up to the first one which becomes greater than $r^*(k)$. For each value of k, the value of $r^*(k)$ is shown in numbers.

Whenever r_i is greater than $r^*(k)$, if **EQ** yields k discrepancies at iteration i then $\mathcal{A}_i$ could be accepted as being ϵ-approximately correct with respect to $\mathcal{N}$ with confidence at least $1-\delta$. For values of r_i smaller than $r^*(k)$, **EQ** *must* yield a number of discrepancies *smaller* than k for the automaton to be accepted as ϵ-approximately correct.

The diagonal line in Fig. 3 is a linear regression that shows the evolution of r^* as a function of k. Notice that the value of r^* seems to increase linearly with k. However, if we look at the increments, we observe that this is not the case. As Fig. 4 shows, they most likely exhibit a log-like growth.

Sample size revisited. The previous observations suggest that it could be possible to cope with an a-priori given number k of acceptable discrepancies in the **EQ** test by taking larger samples. This could be done by revisiting the formula (1) to compute sample sizes by introducing k as parameter of the algorithm.

Following Angluin's approach, let us take δ_i to be $2^{-i}\delta$. We want to ensure:

$$\mathbf{P}(K_{r_i} = k \mid \mathbf{P}(\mathcal{A}_o \oplus \mathcal{A}_t) > \epsilon) < \binom{r_i}{k} e^{-\epsilon(r_i - k)} < 2^{-i}\delta$$

Hence, the smallest such r_i is:

$$r_i = \arg\min_{r \in \mathbb{N}} \left\{ \ln \binom{r}{k} - \epsilon(r-k) + i \ln 2 - \ln \delta < 0 \right\} \tag{5}$$

Notice that for $k = 0$, this gives the same sample size as in equation (1).

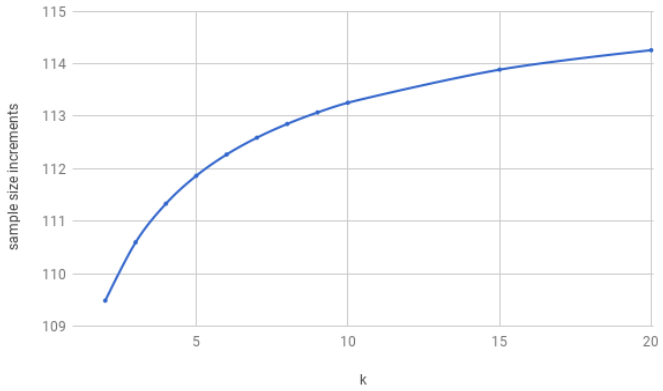


Fig. 4. Increments of r^* as function of k

Now, with sample size r_i, we can ensure a total confidence of $1 - \delta$:

$$\sum_{i>0} \mathbf{P}(K_{r_i} = k \mid \mathbf{P}(\mathcal{A}_o \oplus \mathcal{A}_t) > \epsilon) < \sum_{i>0} 2^{-i}\delta < \delta$$

Although computing the sample size at each iteration using equation (5) does allow to cope with up to a given number of discrepancies k in the **EQ** test, it does not ensure termination. Moreover, solving equation (5) is computationally expensive, and changing the **EQ** test so as to accepting at most k divergences in a set of r_i samples guarantees the same confidence and approximation as passing the standard zero-divergence test proposed in PAC. Hence, in this work we compute r_i as in equation (1). Then, we analyze the values of ϵ_i^* that result *after* stopping Bounded-L^* when a complexity constraint has been reached, and compare them with ϵ and measured errors on a test set.

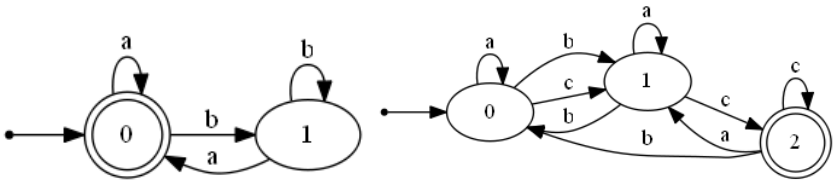
Remark One could argue that this analysis may be replaced by running the algorithm with less restrictive bounds. The main issue here is that the target concept (the language of the ANN) may not be in the hypothesis space (regular languages). Thus, there is no guarantee a hypothesis exists for which the PAC condition holds for whichever ϵ and δ. So, even if the user fixed looser parameters, there is no guarantee the algorithm ends up producing a PAC-conforming DFA.

4 Experimental results

We implemented Bounded-L^* and applied it to several examples. In the experiments we used LSTM networks. Two-phase early stopping was used to train the LSTM, with an 80-20% random split for train-test of a randomly generated dataset. For evaluating the percentage of sample sequences in the symmetric difference we used 20 randomly generated datasets. The LSTM were trained with

sample datasets from known automata as a way of validating the approach. However it is important to remark that in real application scenarios such automata are unknown, or the dataset may not come from a regular language.

Example 1. Let us consider the language $(a+b)^*a+\lambda$, with $\Sigma = \{a, b\}$ and λ being the empty sequence (Fig. 5a). We performed 100 runs of the algorithm with different values for the ϵ and δ parameters. For each run Bounded-L^* terminated normally and obtained a DFA that was an ϵ-approximation of the neural network with confidence at least $1-\delta$. Actually, every learned automata was equivalent to the original automaton and had no differences in the evaluation of the test datasets with regards to the neural network. □



(a) Automaton of Exa. 1 (b) Automaton of Exa. 2

Fig. 5. Automata examples

The previous experiment illustrates that when the neural network is well trained, meaning that it exhibits zero error with respect to all test datasets, the learner will, with high probability, learn the original automaton, which is unknown to both teacher and learner. This case would be the equivalent to using the automaton as the **MQ** oracle, falling in the setting of PAC based L^*. This situation rarely happens in reality, as neural networks, or any model, are never trained to perfectly fit the data. Therefore we are interested in testing the approach with neural networks that do not perfectly characterize the data.

Example 2. Consider the DFA shown in Fig. 5b, borrowed from [26]. The training set contained 160K sequences of variable length up to a maximum length of 10. The error measured on another randomly generated sample test set of 16K sequences between the trained LSTM and the DFA was 0.4296. That is, the LSTM does not perform very well: what language did it actually learn?

Bounded-L^* was executed 20 times with $\epsilon = \delta = 0.05$ and a bound of 10 on the number of states. All runs reached the bound and the automaton of the last iteration was the one with smallest error of all iterations.

Fig. 6 summarizes the results obtained. It can be observed that for each run of the experiment, not always the same automaton is reached due to the random nature of the **EQ** oracle sample picking. In the 20 runs, 6 different DFA were produced, identified with letters a to f, with a number of states between 11 and 13 (indicated in parenthesis).

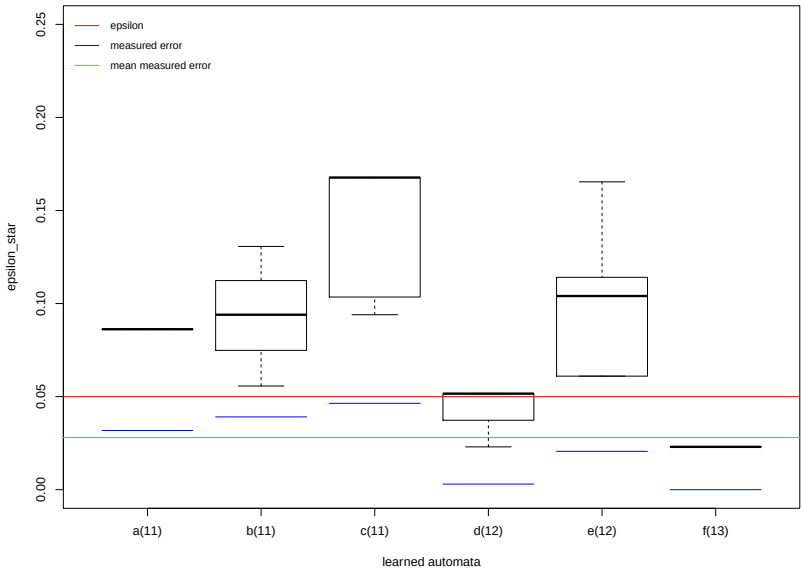


Fig. 6. Measured error, ϵ, and ϵ_i^* for Example 2

For the each automaton, the measured error on the test set is obviously the same. However, ϵ_i^* and δ_i^* may differ, because they depend on the number of iterations and on the number of discrepancies on the randomly generated sample sets used by oracle **EQ** in each case. This is depicted in the figure with a boxplot of the ϵ_i^* values for each automaton produced. It is important to notice that the percentage of sequences in the symmetric difference (measured error) between each learned automaton and the neural network, measured on a set of randomly generated sample test sets, is always below the calculated ϵ_i^* value. It means that the measured empirical errors are consistent with the theoretical values. It is interesting to observe that the measured error was smaller than ϵ. □

Example 3. This example presents the results obtained with an LSTM trained with a different dataset with positive and negative sequences of the same automaton as the previous example. In this case, the measured error was 0.3346.

We ran Bounded-L^* 20 times with $\epsilon = \delta = 0.05$, and a maximum query length of 12. All runs reached the bound. Fig. 8 summarizes the results obtained. The experiment produced 11 different DFA, named with letters from a to k, with sizes between 4 and 27 number of states (indicated in parenthesis). Fig. 7 shows the 12-state automaton with smallest error. All automata exhibited measured errors smaller or equal than $\epsilon_i^\star$. In contrast, they were all above the proposed ϵ. For automata h (12 states) and j (24 states), measured errors are slightly smaller than the respective minimum ϵ_i^* values.

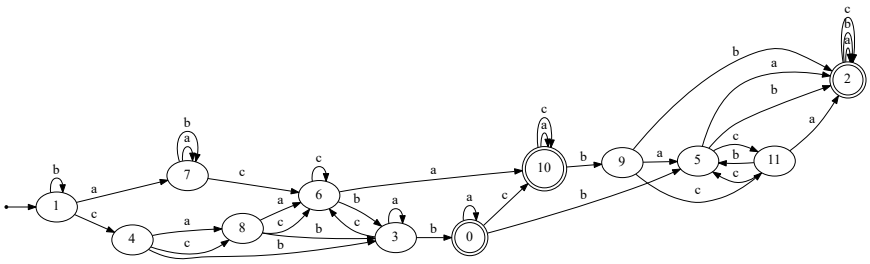


Fig. 7. 12-state automaton with smallest error

Notice that this experiment shows higher variance in the number of automata, number of automaton states, and $\epsilon^\star$ values than the previous one. □

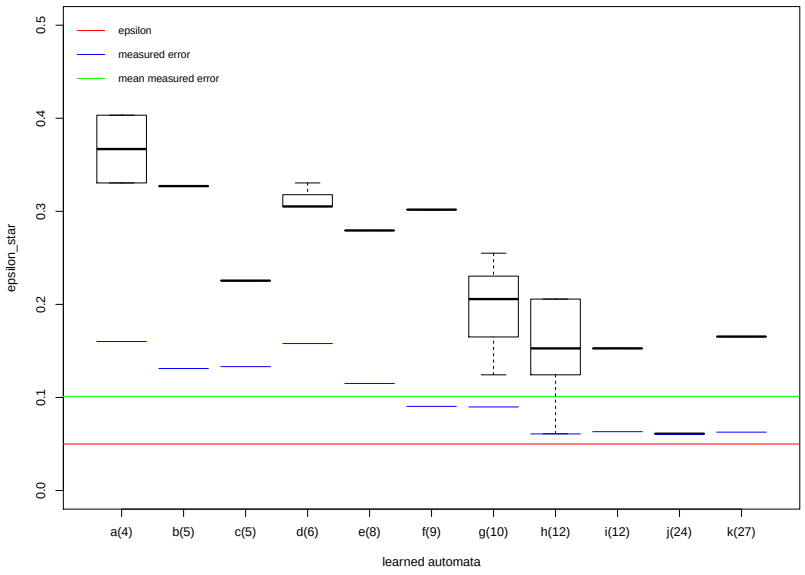


Fig. 8. Measured error, ϵ, and ϵ_i^* of Example 3

Example 4. We study here the Alternating Bit Protocol (Fig. 9). The LSTM measured error was 0.2473. We ran Bounded-L^* with $\epsilon = \delta = 0.05$, and a maximum query length of 5.

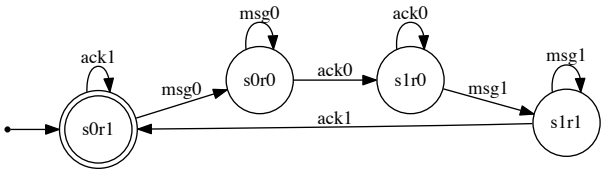


Fig. 9. Alternating bit protocol automaton

Two different automata were obtained, named a and b, with 2 and 4 states respectively (Fig. 11). All runs that produced a reached the bound. This is explained in Fig. 10 where the boxplot for a is above ϵ. On the other hand, almost all runs that produced b completed without reaching the bound. Nevertheless, in all cases where the bound was reached, the sample size was big enough to guarantee $\epsilon^\star$ to be smaller than ϵ. Overall, the empirical error measures were consistent with the theoretical values. □

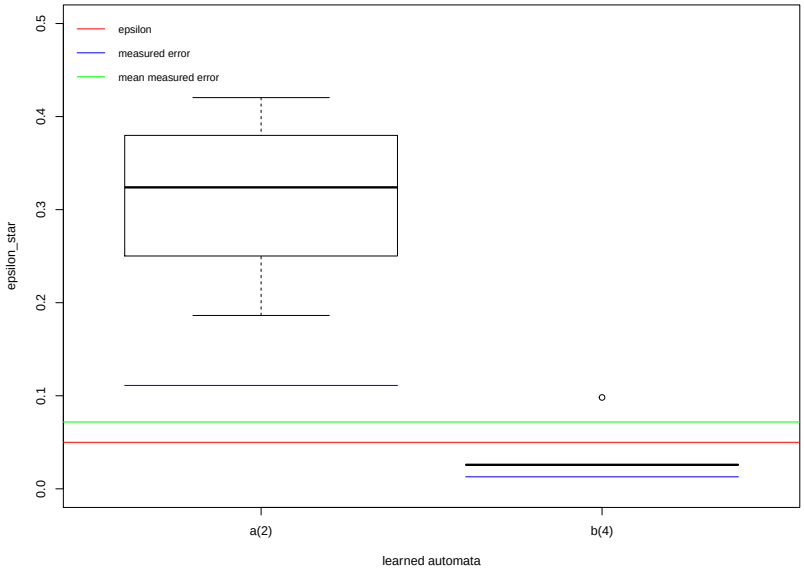
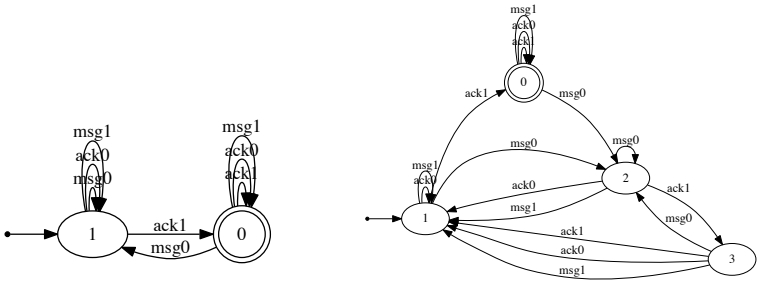


Fig. 10. Measured error, ϵ, and ϵ_i^* of Exa. 4



(a) 2-state automaton (b) 4-state automaton

Fig. 11. Generated automata for ABP

Example 5. We consider here an adaptation of the e-commerce website presented in [24] (Fig. 12). The labels are explained in the following table:

os: open session	ds: destroy session
gAP: get available product	eSC: empty shopping cart
gSC: get shopping cart	aPSC: add product to shopping cart
bPSC: buy products in shopping cart	

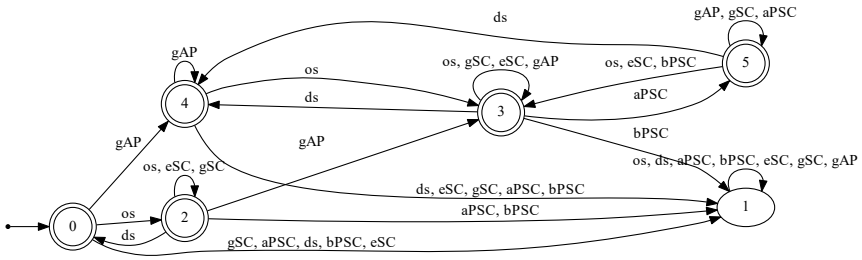


Fig. 12. Model of the e-commerce example adaptation

The training set contained 44K sequences up to a maximum length of 16. The test set contained 16K sequences. The measured error of the ANN on the test set was 0.0000625. We ran Bounded-L^* with $\epsilon = \delta = 0.05$, a maximum query length of 16 and a bound of 10 on the number of states. Fig. 13 shows the experimental results.

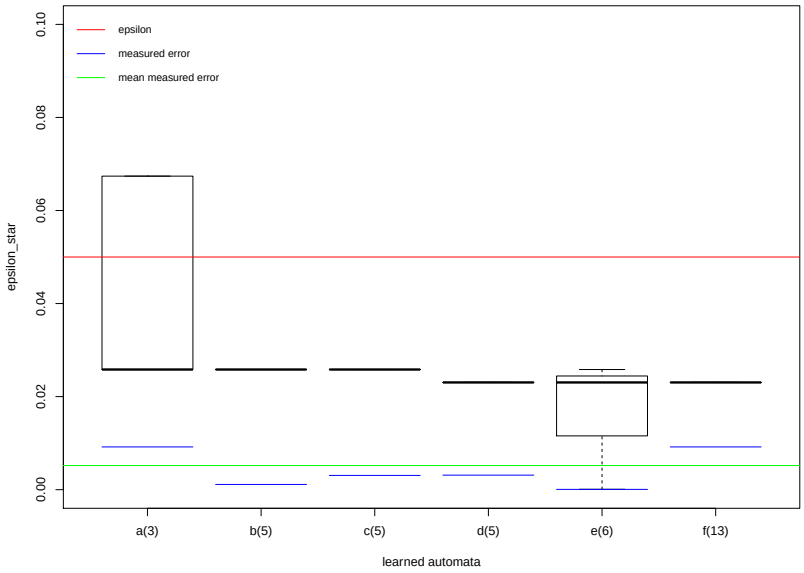


Fig. 13. Measured error, ϵ, and ϵ_i^* of Example 5

Fig. 14 shows one of the learned automata. It helps interpreting the behavior of the network. For example, given $oS, gAP, aPSC, aPSC, bPSC$, the output of the network is 1, meaning that the sequence is a valid sequence given the concept the network was trained to learn. Besides, this sequence also accepted by the automaton, yielding a traceable way of interpreting the result of the network. Moreover, for the input sequence $oS, gAP, aPSC, gSC, eSC, gAP, gAP, bPSC$, we find that the network outputs 1. However, this sequence is not accepted by the automaton (like the majority of the learned models). This highlights that the network misses an important property of the e-commerce site workflow: it is not possible to buy products when the shopping cart is empty. □

Remark The variance on the outcomes produced by Bounded-L^* for the same input ANN could be explained by representational, statistical and computational issues [7]. The first occurs because the language of the network may not be in the hypothesis space, due to the fact that ANN are strictly more expressive than DFA. The second and third are consequences of the sampling performed by **EQ** and the policy used to choose the counter-example.

5 Related Work

A thorough review of the state of the art in explainable AI is presented in [10,14].

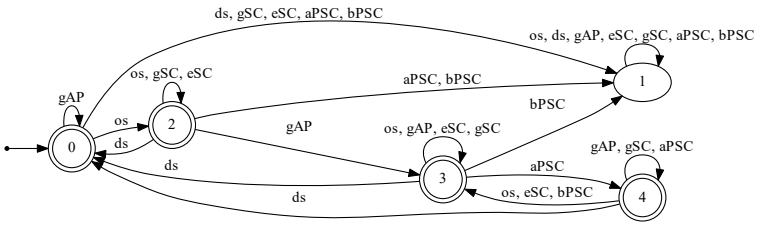


Fig. 14. One of the learned models of the e-commerce example adaptation

To the best of our knowledge, the closest related works to ours are the following. The approaches discussed in [3] are devoted to extracting decision trees and rules for specific classes of multi-layer feed-forward ANN. Besides, such models are less expressive than DFA. Approaches that aim at extracting DFA are white box, that is, they rely on knowing the internal structure of the ANN. For instance, [9] deals with second-order RNN. The algorithm developed in [31] proposes an equivalence query based on the comparison of the proposed hypotheses with an abstract representation of the RNN that is obtained through an exploration of its internal state.

Work on regular inference [15] focused on studying the learnability of different classes of automata but none was applied to extracting them from ANN.

None of these works provide means for black-box model explanation in the context of ANN. Moreover, our approach using PAC regular inference is completely agnostic of the model.

6 Conclusions

We presented an active PAC-learning algorithm for learning automata that are approximately correct with respect to neural networks. Our algorithm is a variant of Angluin's L^* where a bound on the number of states or the query length is set to guarantee termination in application domains where the language to be learned may not be a regular one. We also studied the error and confidence of the hypotheses obtained when the algorithm stops by reaching a complexity bound.

The experimental evaluation of our implementation showed that the approach is able to infer automata that are reasonable approximations of the target models with high confidence, even if the output model does not pass the usual 0-divergence **EQ** statistical test of the PAC framework. These evaluations also provided empirical evidence that the method exhibits high variability in the proposed output models. This is a key concern to be addressed in future work.

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_06`.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (Nov 1987)
2. 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)
3. Bologna, G., Hayashi, Y.: Characterization of symbolic rules embedded in deep dimlp networks: A challenge to transparency of deep learning. Journal of Artificial Intelligence and Soft Computing Research **7**(4), 265–286 (2017)
4. Calderon, T.G., Cheh, J.J.: A roadmap for future neural networks research in auditing and risk assessment. International Journal of Accounting Information Systems **3**(4), 203 – 236 (2002), second International Research Symposium on Accounting Information Systems
5. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)
6. Deng, L., Chen, J.: Sequence classification using the high-level features extracted from deep neural networks. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 6844–6848 (May 2014)
7. Dietterich, T.G.: Ensemble methods in machine learning. In: Proceedings of the First International Workshop on Multiple Classifier Systems. pp. 1–15. MCS '00, Springer-Verlag, London, UK, UK (2000)
8. Gers, F.A., Schmidhuber, E.: Lstm recurrent networks learn simple context-free and context-sensitive languages. IEEE Transactions on Neural Networks **12**(6), 1333–1340 (Nov 2001)
9. Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., Lee, Y.C.: Learning and extracting finite state automata with second-order recurrent neural networks. Neural Comput. **4**(3), 393–405 (May 1992)
10. Gilpin, L.H., Bau, D., Yuan, B.Z., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: An approach to evaluating interpretability of machine learning. CoRR **abs/1806.00069** (2018)
11. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3), 302 – 320 (1978)
12. Gorman, K., Sproat, R.: Minimally supervised number normalization. TACL **4**, 507–519 (2016)
13. Grzes, M., Taylor, M. (eds.): Proceedings of the Adaptive and Learning Agents Workshop (2010)
14. Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., Giannotti, F.: A survey of methods for explaining black box models. CoRR **abs/1802.01933** (2018)
15. 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)
16. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997)
18. Holzinger, A., Biemann, C., Pattichis, C.S., Kell, D.B.: What do we need to build explainable ai systems for the medical domain? arXiv:1712.09923 (2017)

19. Holzinger, A., Plass, M., Holzinger, K., Crisan, G.C., Pintea, C.M., Palade, V.: A glass-box interactive machine learning approach for solving np-hard problems with the human-in-the-loop. arXiv:1708.01104 (2017)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436 – 444 (2015)
21. Lei, T., Barzilay, R., Jaakkola, T.: Rationalizing neural predictions. In: Empirical Methods in Natural Language Processing (EMNLP) (2016)
22. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: Proceedings of 23rd European Symp. on Artificial Neural Networks, Computational Intelligence and Machine Learning (2015)
23. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (Dec 1943)
24. Merten, M.: Active automata learning for real life applications (01 2013)
25. Murphy, K.: Passively learning finite automata. Tech. Rep. 96-04-017, Santa Fe Institute (1996)
26. Omlin, C.W., Giles, C.L.: Constructing deterministic finite-state automata in recurrent neural networks. J. ACM **43**(6), 937–972 (Nov 1996)
27. 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, April 19-24, 2015. pp. 1916–1920 (2015)
28. Sarikaya, R., Hinton, G.E., Deoras, A.: Application of deep belief networks for natural language understanding. IEEE/ACM Transactions on Audio, Speech, and Language Processing **22**(4), 778–784 (April 2014)
29. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (Nov 1984)
30. Wang, Y., Tian, F.: Recurrent residual learning for sequence classification. In: Proc. 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016. pp. 938–943 (2016)
31. 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)
32. Xing, Z., Pei, J., Keogh, E.: A brief survey on sequence classification. SIGKDD Explor. Newsl. **12**(1), 40–48 (Nov 2010)
33. Zhang, C., Jiang, J., Kamel, M.: Intrusion detection using hierarchical neural networks. Pattern Recognition Letters **26**(6), 779 – 791 (2005)
34. Zhou, C., Cule, B., Goethals, B.: Pattern based sequence classification. IEEE Transactions on Knowledge and Data Engineering **28**(5), 1285–1298 (May 2016)
35. Zhou, C., Sun, C., Liu, Z., Lau, F.C.M.: A C-LSTM neural network for text classification. CoRR **abs/1511.08630** (2015)