Passive Learning of Lattice Automata from Recurrent Neural Networks

Jaouhar Slimi, Tristan Le Gall and Augustin Lemesle

Université Paris-Saclay, CEA, List, France

Abstract

We present a passive automata learning algorithm that can extract automata from recurrent networks with very large or even infinite alphabets. Our method combines overapproximations from the field of Abstract Interpretation and passive automata learning from the field of Grammatical Inference. We evaluate our algorithm by first comparing it with the state-of-the-art automata extraction algorithm from Recurrent Neural Networks trained on Tomita grammars. Then, we extend these experiments to regular languages with infinite alphabets, which we propose as a novel benchmark.

Keywords

Automata Learning, Recurrent Neural Networks, Abstract Interpretation

1. Introduction

Recurrent Neural Networks (RNNs) are a class of Deep Neural Networks conceived primarily to process sequential data. Their linear complexity in space and time, combined with their efficiency, leads to their adoption in various applications from Time Series Forecasting [1] to policy modelling in reinforcement learning environments [2]. RNNs have been extensively studied from the lens of automata and formal language theory to model and analyze their behavior [3, 4]. Concretely, automata learning from RNNs seeks to map (or approximate) its semantics to some kind of automaton.

Automata learning methods are either based on active learning, where a form of feedback is received from the System Under Learning (SUL), which is the RNN in our case, or passive learning, where we only require a set of execution traces. Our work focuses on the latter approach. Our goal is to construct a symbolic automaton; more precisely, a Lattice Automaton, from a set of traces of an RNN. Lattice Automata are similar to Deterministic Finite Automata (DFA), however, they can recognize languages over an *infinite alphabet*, such as the inputs of an RNN, which are usually in the form of floating-point vectors. Therefore, we can be closer to its real behavior, and give a surrogate model of an RNN trained to recognize a series of real-world inputs (e.g. language modelling tasks such as sentiment analysis [5]).

Outline: We first briefly present a few key notions needed to understand our work. Then, we present our technique and some experimental results. We conclude with the advantages and limitations of this approach, and our future work.

Related work: In recent years, there has been a renewed interest in learning automata from RNNs, for interpretability as well as for verification purposes. Active learning techniques are mainly based on extensions of the L* algorithm for learning regular languages. Multiple works have been proposed [6, 7, 8] to extend L* for extracting automata from RNNs.

On the other hand, passive learning techniques rely on the hypothesis that in RNNs, semantically similar intermediate neurons (also called hidden states) tend to cluster together [9].

OVERLAY 2025: 7th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, October 26th, Bologna, Italy

jaouhar.slimi@cea.fr (J. Slimi); tristan.le-gall@cea.fr (T. Le Gall); augustin.lemesle@cea.fr (A. Lemesle)

Multiple techniques relying on clustering were proposed for learning DFA [10, 11], Weighted Finite Automata (WFA) [12], Hidden Markov Models (HMM) [13], and Probabilistic Finite Automata (PFA) [14]. The type of automata influences the complexity of the solution and is mainly motivated by the application; for instance, DFAs are well-suited to deterministic RNN behaviors, whereas WFAs and PFA are better alternatives to capture their probabilistic dynamics. A different approach by [15] relies on the classical Gold algorithm [16] by creating a prefix tree automaton (PTA) from the execution traces, then merging its states to yield a DFA. The authors associate RNNs hidden states values to each state of the PTA and then use that during the merge algorithm by only merging states that have at most a distance d between their associated hidden states. In this paper, we adapt the latter method to learn automata over an infinite alphabet.

2. Key Notions

Elman Networks A category of simple RNN architectures, defined by a function $F_\theta(x_t, h_{t-1})$ that computes the hidden state h_t, followed by a classifier function $G_\phi(h_t)$ that computes the output y_t. F_θ and G_ϕ are non linear functions parameterized by their weights θ and ϕ. For readability purposes, we assume that both the input vector x_t and the hidden state h_{t-1} are vectors in $\mathbb{R}^d$, at any time step $t = 1, 2 \dots n$ [1]. In our paper, for simplicity, we consider only binary classification networks, *i.e.* $y_t \in \mathbb{B}$. The computation carried out by the RNN after processing an input vector x_t is then in the following form: $(x_t, h_{t-1}) \xrightarrow{F_\theta} h_t \xrightarrow{G_\phi} y_t$.

Abstract Lattice Let us consider finite sequences on $\mathbb{R}^d$ (which is our infinite alphabet). We want to abstract $(\mathcal{P}(\mathbb{R}^d), \subseteq)$ using a lattice $(\Lambda, \sqsubseteq)$, which is defined as follows:

- A relation $\sqsubseteq$ on a set Λ is a *partial order* if:
  1. $\forall \lambda \in \Lambda, \lambda \sqsubseteq \lambda$
  2. $\forall \lambda_1, \lambda_2 \in \Lambda, (\lambda_1 \sqsubseteq \lambda_2) \wedge (\lambda_2 \sqsubseteq \lambda_1) \Rightarrow \lambda_1 = \lambda_2$
  3. $\forall \lambda_1, \lambda_2, \lambda_3 \in \Lambda, (\lambda_1 \sqsubseteq \lambda_2) \wedge (\lambda_2 \sqsubseteq \lambda_3) \Rightarrow \lambda_1 \sqsubseteq \lambda_3$
- $(\Lambda, \sqsubseteq)$ is a *complete lattice* if:
  1. $\sqsubseteq$ is a *partial order* on a set Λ
  2. any subset of $\Omega \subseteq \Lambda$ has a greatest lower bound $\sqcap\Omega$ (*i.e.* the set $\{y \in \Lambda \,|\, \forall x \in \Omega, x \sqsupseteq y\}$ has a greatest element), and a least upper bound $\sqcup\Omega$ (*i.e.* the set $\{y \in \Lambda \,|\, \forall x \in \Omega, x \sqsubseteq y\}$ has a smallest element)
- Then we define:
  - $\bot = \sqcup\emptyset = \sqcap\Lambda$
  - $\top = \sqcap\emptyset = \sqcup\Lambda$
  - $\mathrm{Atoms}(\Lambda)$ as the set of the minimal elements of $\Lambda \setminus \bot$; in other words, $a \in \Lambda$ is an atom if $\forall \lambda \in \lambda, \lambda \sqsubseteq a \Rightarrow \lambda = a \vee \lambda = \bot$

Atomistic Lattice A lattice $(\Lambda, \sqsubseteq)$ is *atomistic* if: $\forall \lambda \in \Lambda : \lambda = \sqcup\{a \in \mathrm{Atoms}(\Lambda) \,|\, a \sqsubseteq \lambda\}$. Therefore, we can define a finite partition of $\mathrm{Atoms}(\Lambda)$ either as we did previously, or as a function $\Pi : \{1 \dots n\} \to \Lambda$ satisfying the following property: $\forall a \in \mathrm{Atoms}(\Lambda)$ there exists a unique $i \in \{1 \dots n\}$ such that $a \sqsubseteq \Pi(i)$. In this paper, and for the sake of simplicity, $(\Lambda, \sqsubseteq)$ is defined by the box abstraction.

[1] if the dimensions are not the same, we can always resize them to the highest dimension

The Box Abstraction Any set $\Omega \subseteq \mathbb{R}^d$ can be abstracted by a box (d-uple of intervals) $\alpha(\Omega) = \langle \omega_1 \dots \omega_d \rangle$, such that each ω_i is an interval bounding the projection of Ω on the dimension i. It is a classical abstract domain, and there is a Galois connection [17] between $(\mathcal{P}(\mathbb{R}^d), \subseteq)$ and $(\Lambda, \sqsubseteq)$. This lattice is atomistic: its atoms are the singletons $\langle [\![x_1, x_1]\!] \dots [\![x_d, x_d]\!] \rangle$ for any vector $\langle x_1 \dots x_d \rangle \in \mathbb{R}^d$, thus the isomorphism between $\mathrm{Atoms}(\Lambda)$ and $\mathbb{R}^d$. Because of this isomorphism, we identify any $x \in \mathbb{R}^d$ with $\alpha(x) \in \Lambda$. That is why, in the definition of the language recognized by a Lattice Automaton, we wrote the condition "$x_i \sqsubseteq \lambda_i$", while strictly speaking, the condition should be "$\alpha(x_i) \sqsubseteq \lambda_i$" since $x_i \in \mathbb{R}^d$.

Partition A finite partition on $\mathrm{Atoms}(\Lambda)$ is a function $\Pi : \{1 \dots n\} \to \mathcal{P}(\mathrm{Atoms}(\Lambda))$ such that $\bigcup_{i=1}^{n} \Pi(i) = \mathrm{Atoms}(\Lambda)$ and $i \neq j \Rightarrow \Pi(i) \cap \Pi(j) = \emptyset$. Since Λ is atomistic, for any $i \in \{1 \dots n\}$, we can identify the set of atoms $\Pi(i)$ with $\sqcup\{a | a \in \Pi(i)\}$ which defines the "maximal element" (also noted $\Pi(i)$ by abuse of notation) of this class. We have the property that classes of the partition are stable for the operation $\sqcup$: $\forall \lambda_1, \lambda_2 : \lambda_1 \sqsubseteq \Pi(i) \wedge \lambda_2 \sqsubseteq \Pi(i) \Rightarrow \lambda_1 \sqcup \lambda_2 \sqsubseteq \Pi(i)$. Partitions are needed to properly define Lattice Automata, see [18] for details.

Lattice Automata Lattice Automata are similar to DFA, but with transitions labelled by the elements of the atomistic lattice Λ rather than elements of a finite alphabet Σ. Formally, a Lattice Automaton is a tuple $\mathcal{A} = \langle \Lambda, \Pi, Q, \mathbb{I}, \mathbb{F}, \delta, \Gamma \rangle$ such that:

- Λ is an atomistic lattice
- Π is a partition of $\mathrm{Atoms}(\Lambda)$
- Q is a finite set of states
- $\mathbb{I} \subseteq Q$ are the initial states
- $\mathbb{F} \subseteq Q$ are the final states
- $\delta \subseteq Q \times \Lambda \times Q$ is the set of transitions
- $\Gamma : Q \to \Lambda$ is the hidden states function

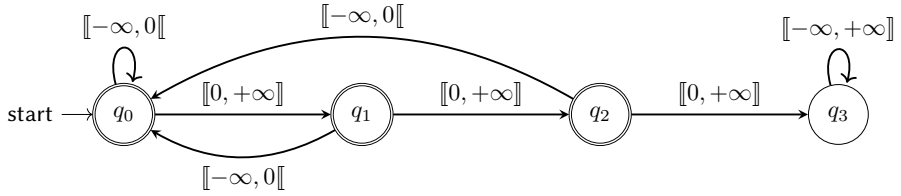

Figure 1: Example of a Lattice Automaton. $\mathbb{I} = \{q_0\},\ \mathbb{F} = \{q_0, q_1, q_2\}$.

We also require the following two properties to ensure that the number of transitions is finite, even if the alphabet is infinite:

1. For any transition $(q, \lambda, q') \in \delta$, there is $i \in \{1 \dots n\}$, such that: $\forall a \in \mathrm{Atoms}(\Lambda), a \sqsubseteq \lambda \Rightarrow a \in \Pi(i)$. In other words, a transition shall not mix atoms belonging to different classes of the partition. Thus, we can define a function $\Pi^{-1} : \Lambda \to \{1 \dots n\}$ that associates a label λ of a transition to its class of the partition.
2. For any couple of states (q, q') and any class of the partition $i \in \{1 \dots n\}$, there is at most one transition $(q, \lambda, q) \in \delta$ such that $\Pi^{-1}(\lambda) = i$.

In this work, since in our examples and experiments $(\Lambda, \sqsubseteq)$ we rely on the box abstraction, we will simply call them Interval Lattice Automata (ILA).

Language recognition Like a DFA, an ILA recognizes a language, *i.e.* a set of words on the input alphabet $\mathrm{Atoms}(\Lambda) \equiv \mathbb{R}^d$. A word $x_1 \dots x_k$ ($x_i \in \mathbb{R}^d$ for all i) is accepted by $\mathcal{A}$ if there is a sequence $q_0 \xrightarrow{\lambda_1} q_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_k} q_k$ such that:

- $\forall\, i = \{1 \dots k\}, (q_{i-1}, \lambda_i, q_i) \in \delta$ and $x_i \sqsubseteq \lambda_i$
- $q_0 \in \mathbb{I}$ and $q_k \in \mathbb{F}$

Representation of the hidden states The function Γ does not play any role in the definition of the language recognized by an ILA. Its purpose is to associate any state of the automaton with a set of hidden states of the RNN, abstracted by an element of Λ. Unlike the elements labelling the transitions, we do not require $\Gamma(q)$ to belong to a single class of the partition Π, it can be anything (including $\top$). We included this function in the definition of the automaton since it has a role in the learning algorithm.

Adding transitions Algorithms on ILA are similar to the ones on DFA. However, one must ensure that the two properties stated above remain true. For example, when we want to add a transition (q, λ, q') to an ILA, we must first check if there already exists a transition between the two states q and q', labelled by λ' belonging to the same class as λ. If so, that transition is replaced by $\lambda \sqcup \lambda'$, as written in Algorithm 1.

Algorithm 1 Add Transition

Input:
An automaton $\mathcal{A} = \langle \Lambda, \Pi, Q, \mathbb{I}, \mathbb{F}, \delta, \Gamma \rangle$ and $(q, \lambda, q') \in Q \times \Lambda \times Q$ such that $\exists i \in \{1 \dots n\} \lambda \sqsubseteq \Pi(i)$
Output:
$\mathcal{A}_m = \langle \Lambda, \Pi, Q_m, \mathbb{I}_m, \mathbb{F}_m, \delta_m, \Gamma_m \rangle$
Initialize:
$Q_m \leftarrow Q$
$\mathbb{I}_m \leftarrow \mathbb{I}$
$\mathbb{F}_m \leftarrow \mathbb{F}$
$\delta_m \leftarrow \delta$
$\Gamma_m \leftarrow \Gamma$
$j \leftarrow \Pi^{-1}(\lambda)$
if $\exists (q, \lambda', q') \in \delta_m$ such that $\Pi^{-1}(\lambda') = j$ **then**
    $\delta_m \leftarrow \delta_m \setminus \{(q, \lambda', q')\} \cup \{(q, \lambda \sqcup \lambda', q')\}$;
else
    $\delta_m \leftarrow \delta_m \cup \{(q, \lambda, q')\}$;
end if
return $\mathcal{A}_m$;

3. Learning Lattice Automata from RNN execution traces

The proposed algorithm is an extension of Gold's algorithm for the case of ILA. We first create an Interval Prefix Tree Automaton (IPTA) from a set of traces of the RNN. Then, we launch the merging phase in which we merge states according to a *similarity score* inspired by [15].

Building the Interval Prefix Tree Automaton (IPTA) Let us consider a finite set of RNN execution traces $S = \{s_i = \left(x_1^{(i)}, h_1^{(i)}, y_1^{(i)}\right) \dots \left(x_{k_i}^{(i)}, h_{k_i}^{(i)}, y_{k_i}^{(i)}\right) \mid k_i \geq 1\}$, where sequences s_i are of varying lengths k_i. We build the IPTA starting with an automaton with a single initial state q_0 and applying the function Add Sequence$(\mathcal{A}, s)$ for every sequence $s \in S$. The function Add Sequence$(\mathcal{A}, s)$ takes a sequence $s = (x_1, h_1, y_1), (x_2, h_2, y_2) \dots (x_k, h_k, y_k)$ and adds transitions and states to the automaton $\mathcal{A}$ in the following way:

We consider the initial state q_0 and the first triplet (x_1, h_1, y_1) of the sequence:

- If there exists already in $\mathcal{A}$ a state q_1 and a transition (q_0, λ_1, q_1) such that $\Pi^{-1}(\lambda_1) = \Pi^{-1}(x_1)$, then we modify the transition (q_0, λ_1', q_1); with $\lambda' = \lambda \sqcup [\![x_1, x_1]\!]$, and the Γ function to $\Gamma(q_1) \leftarrow \Gamma(q_1) \sqcup [\![h_1, h_1]\!]$; moreover, q_1 becomes a final state if $y_1 = 1$

- otherwise, we create a new state q_1 and a transition (q_0, x_1, q_1), with $\Gamma(q_1) \leftarrow h_1$ and q_1 being a final state if $y_1 = 1$

The process is then repeated with q_1 and the second triplet (x_2, h_2, y_2), and so on, until the end of the sequence. This construction ensures that any word $s = (x_1, h_1, y_1), (x_2, h_2, y_2) \dots (x_k, h_k, y_k)$ such that $y_k = 1$ will also be accepted by the IPTA, and that $[\![h_k, h_k]\!] \sqsubseteq \Gamma(q_k)$. For an example, Figure 2 shows the resulting IPTA from the set of traces S:

$$S = \left\{ \begin{array}{rcl} s_1 & = & (1.4, h_1^1, 1),\ (-1.07, h_1^2, 1),\ (1.08, h_1^3, 1),\ (-7.06, h_1^4, 1),\ (9.03, h_1^5, 1),\ \dots \\ s_2 & = & (3.39, h_2^1, 1),\ (-3.2, h_2^2, 1),\ (7.91, h_2^3, 1),\ (-3.45, h_2^4, 1),\ (2.1, h_2^5, 1),\ \dots \\ s_3 & = & (1.9, h_3^1, 1),\ (3.56, h_3^2, 1),\ (3.14, h_3^3, 0),\ (-33.2, h_3^4, 0),\ \dots \\ s_4 & = & (2.3, h_4^1, 1),\ (2.29, h_4^2, 1),\ (2.06, h_4^3, 0),\ (-0.51, h_4^4, 0) \\ \dots & & \end{array} \right.$$

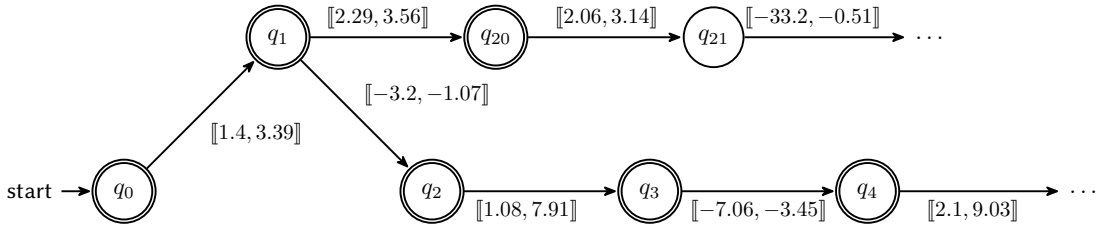

Figure 2: Example of an IPTA created given the set S from a RNN trained on language 4 of Tomita 2.0

As shown in the example, there are already some merging involved when building the IPTA, since, for each state of the IPTA, there are at most n outgoing transitions (one for each class of the partition). Consequently, the language accepted by the IPTA is larger than the set of traces S, and there is the implicit assumption that the set of traces is coherent w.r.t. the chosen partition Π. It can be checked by verifying that there cannot be two sequences $s = (x_1, h_1, y_1) \dots (x_k, h_k, y_k)$ and $s' = (x'_1, h'_1, y'_1) \dots (x'_k, h'_k, y'_k)$ such that: s and s' are prefixes of sequences of S, $\forall\, 1 \leq i \leq k,\ \Pi^{-1}(x_i) = \Pi^{-1}(x'_i)$, and $y_k \neq y'_k$.

If that property does not hold, it means the partition is too coarse to even build the IPTA, and that our method cannot yield a faithful representation of the behavior of the RNN. In that case, we can try again with a finer partition.

Merging the states The second step of the algorithm is to merge states according to their similarity score (a real number between 0 and 2), as long as it is possible. A description of our method is described in algorithm 2.

In our examples and experiments, the similarity score of two states q_1 and q_2 is defined as follows:

- If only one of the two states q_1 and q_2 belongs to $\mathbb{F}$, then the score is 2
- If both states (or none) belong(s) to $\mathbb{F}$, then the score is:

$$\textsc{Similarity score}(q_i, q_j) = 1 - \cos\left(mid(\Gamma(q_i)), mid(\Gamma(q_i))\right) = 1 - \frac{mid(\Gamma(q_i)) \cdot mid(\Gamma(q_j))}{\|mid(\Gamma(q_i))\|_2 \, \|mid(\Gamma(q_j))\|_2}$$

where $mid(\Gamma(q_i))$ denotes the center of the box $\Gamma(q_i)$.

Therefore, while there is at least one couple of states $q_1 \neq q_2$ such that their similarity score is lower than a hyperparameter d, the two states are merged. It means that any transition that goes to (or originates from) q_1 or q_2 will go to (or originate from) the merged state. In this process, if

Algorithm 2 Merge ILA

Input:
$\mathcal{A} = \langle \Lambda, \Pi, Q, \mathbb{I}, \mathbb{F}, \delta, \Gamma \rangle$, q_i and q_j to be merged
Output:
$\mathcal{A}_m = \langle \Lambda, \Pi, Q_m, \mathbb{I}_m, \mathbb{F}_m, \delta_m, \Gamma_m \rangle$ with q_j merged into q_i
Initialize:
$Q_m \leftarrow Q$
$\mathbb{I}_m \leftarrow \mathbb{I}$
$\mathbb{F}_m \leftarrow \mathbb{F}$
$\delta_m \leftarrow \delta$
$\Gamma_m \leftarrow \Gamma$
for $q \in Q_m$ **do**
  if $\exists (q, \lambda, q_j) \in \delta_m$ **then**
    $\mathcal{A}_m \leftarrow$ ADD TRANSITION$(\mathcal{A}_m, (q, \lambda, q_i))$
  end if
  if $\exists (q_j, \lambda, q) \in \delta_m$ **then**
    $\mathcal{A}_m \leftarrow$ ADD TRANSITION$(\mathcal{A}_m, (q_i, \lambda, q))$
  end if
end for
$\Gamma_m(q_i) \leftarrow \Gamma_m(q_i) \sqcup \Gamma_m(q_j)$
if $q_j \in \mathbb{I}_m$ **then**
  $\mathbb{I}_m \leftarrow \mathbb{I}_m \cup \{q_i\}$;
end if
if $q_j \in \mathbb{F}_m$ **then**
  $\mathbb{F}_m \leftarrow \mathbb{F}_m \cup \{q_i\}$;
end if
$\mathcal{A}_m \leftarrow$ DELETE STATE$(\mathcal{A}_m, q_j)$
return $\mathcal{A}_m$

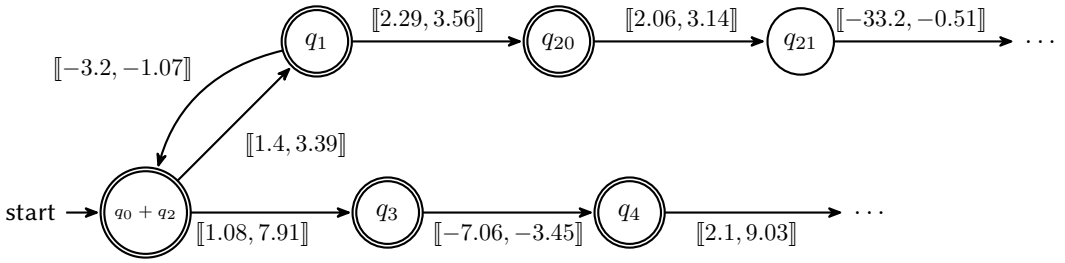

Figure 3: Result of the merging of q_0 and q_2 (from the IPTA)

two transitions (q, λ, q') and (q, λ', q') belong to the same partition class, they will be merged. For example, when we merge the two states q_0 and q_2 of the IPTA depicted in Figure 2, we obtain the ILA depicted in Figure 3.

At the end of the algorithm, we obtain an ILA $\mathcal{A}$ that recognizes a language larger than the set of traces S given as its input. Indeed, if $s = (x_1, h_1, y_1) \dots (x_k, h_k, y_k)$ is a trace in S, then there exists a sequence of states and transitions $q_0 \xrightarrow{\lambda_1} q_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_k} q_k$ in $\mathcal{A}$ such that: $q_0 \in \mathbb{I}$; $\forall\, i = 1 \dots k$ we have $(q_{i-1}, \lambda_i, q_i) \in \delta$, $x_i \sqsubseteq \lambda_i$, if $y_i = 1$ then $q_i \in \mathbb{F}$, and $h_i \sqsubseteq \Gamma(q_i)$.

However, this ILA $\mathcal{A}$ may also accept words that are not accepted by the original RNN, and if the set of traces is too small, it may also reject words that are accepted by the original RNN. It is why we need experimental results to assess the faithfulness of the resulting automaton w.r.t the original RNN.

4. Experiments

Experimental setting Our experiments were run on a Dell Inc. Precision 3591 computer, equipped with an Intel® Core™ Ultra 7 165H × 22 CPU. Python 3.12.3 was used for algorithms implementation and data synthesis tasks. We also used PyRAT [19] to define the abstractions.

We evaluated our algorithm on two benchmarks. First we run experiments with RNNs trained on Tomita languages, to compare our findings with [15] and demonstrate that our approach for inferring ILA is on par with SoTA. The fidelity to the original RNN of ILA are close to DFA but a bit less efficient (Figure 4), and since ILA are more complex than DFA, we have a larger number of states in our ILA. In that sense, our merging approach might benefit from further improvements to reduce the number of states. Then, we propose a novel benchmark by extending the Tomita languages to accept sequences of floats as inputs. For example, the first language accepts only sequences of numbers between 0 and 10. The complete definition is given in Table 1.

Tomita 2.0 language	Language description
1	$([\![0, 10]\!])^\star$
2	$([\![0, 10]\!][\![-10, 0[\!])^\star$
3	No odd $[\![-10, 0[\!]$ after an odd $[\![0, 10]\!]$ string
4	No substring containing 3 consecutive letters $\in [\![-10, 0[\!]$
5	Even number of letters $\in [\![-10, 0[\!]$ and even number of letters $\in [\![0, 10]\!]$
6	Difference of the number of letters $\in [\![0, 10]\!]$ and letters $\in [\![-10, 0[\!]$ is a multiple of 3
7	$[\![-10, 0[\!]^\star[\![0, 10]\!]^\star[\![-10, 0[\!]^\star[\![0, 10]\!]^\star$

Table 1
Definitions of the 7 Tomita 2.0 languages used in the experiments

Tables 2 and 3 summarize our results. Our implementation takes roughly few seconds to infer an automaton from a set of execution traces. The fidelity score is measured as follows $\textsc{Fidelity}(\mathcal{R}, \mathcal{A}, x) = \sum_{i=1}^{n} |\mathcal{R}(x_i) - \mathcal{A}(x_i)|$ for a given RNN $\mathcal{R}$, ILA $\mathcal{A}$ and sequence of inputs x.

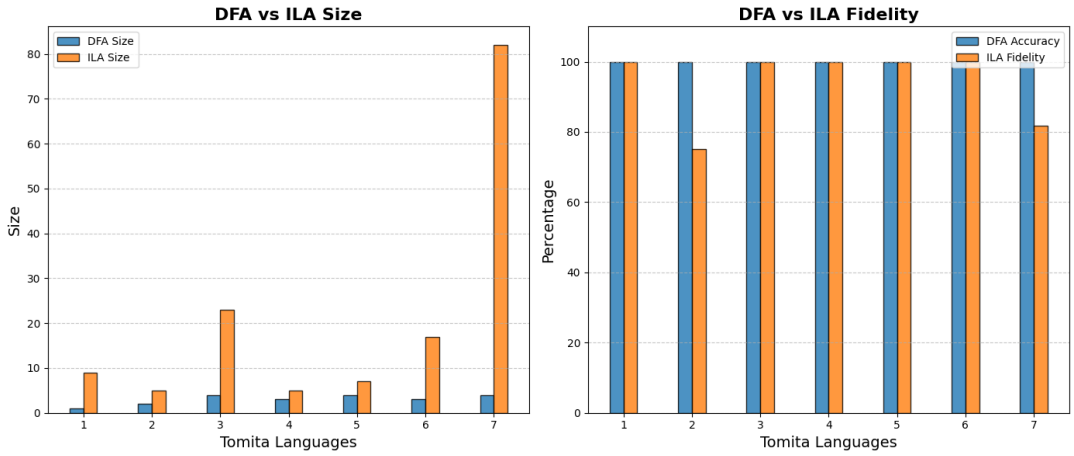

Figure 4: Fidelity and size comparison between ILA and DFA

We also analyze type I and type II errors to evaluate our algorithm:

- A type I error would occur because of abstractions imprecision, when a word is rejected by the RNN and accepted by its automaton (also known as false alarms).
- A type II error reflects failure to capture RNN semantics, which can be due to insufficient sample size used to build the IPTA.

Tomita language	**1**	**2**	**3**	**4**	**5**	**6**	**7**
RNN accuracy	100	100	100	100	100	99.89	100
ILA size	9	5	23	5	7	17	82
ILA Fidelity	100	75.08	100	100	100	100	81.83
Type I error	0	0	0	0	0	0	16.05
Type II error	0	24.92	0	0	0	0	2.12

Table 2
Benchmark results of passive learning of ILA from RNNs trained on Tomita languages

Tomita 2.0 language	**1**	**2**	**3**	**4**	**5**	**6**	**7**
RNN accuracy	99.98	98.64	99.94	98.95	99.89	95.91	96.35
ILA size	10	7	16	43	11	9	40
ILA Fidelity	82.33	62.75	99.14	86.86	88.07	82.75	71.53
Type I error	0.03	0.06	0.01	7.72	0.5	0.02	0
Type II error	17.64	37.19	0.85	5.42	11.43	17.23	28.47

Table 3
Benchmark results of passive learning for ILA from RNNs trained on Tomita 2.0 languages

The scores of extracted ILA from a Tomita 2.0 RNN are roughly 80%, which is acceptable at this stage. The distribution of error scores suggests that our interval abstraction is precise enough (except for language 4, where it can reach 7.72%); however, type II error reflects an insufficient sample size, especially for language 2. Increasing the size of our sequences S (which was set to 1000 sequences of maximum length 20 for each) will likely lead to improving the scores. Our preliminary results can be improved by further benchmarking the merge parameter and analyzing errors. Also by experimenting with different state merging algorithms that perform better than Gold's algorithm, such as RPNI or EDSM [20], would lead to better results.

5. Conclusion and Future Work

We presented in this paper a passive learning algorithm that is capable of inferring an automaton from a set of traces of an RNN. Unlike previous methods, we do not require the inputs of the RNN to belong to a finite alphabet. Our algorithm ensures that we obtain an overapproximation of the set of traces. Our experiments demonstrate a capacity to infer ILA from an RNN trained on possibly infinite regular language, while also indicating that there remains significant potential for further improvements (increase fidelity and reduce ILA size), and to extend our benchmarks.

In future work, we aim to extend this proof of concept, especially for the verification and the explainability of RNNs trained for practical real-world applications, *e.g.* time series forecasting, where the robustness of the RNN is of paramount importance. The automata-based approach is a generalizable formal method offering a synergy between the interpretability and the verification for recurrent networks. While properties that can be formulated and verified are beyond adversarial robustness, the explanation can also be regarded as a property we seek to verify.

Acknowledgements : This work was supported by the French Agence Nationale de la Recherche (ANR) through SAIF (ANR-23-PEIA-0006) as part of the France 2030 programme.

References

[1] S. Lin, W. Lin, W. Wu, F. Zhao, R. Mo, H. Zhang, Segrnn: Segment recurrent neural network for long-term time series forecasting, 2023.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al., Mastering atari, go, chess and shogi by planning with a learned model, Nature 588 (2020) 604–609.

[3] H. Jacobsson, Rule extraction from recurrent neural networks: Ataxonomy and review, Neural Computation 17 (2005) 1223–1263.

[4] B. Bollig, M. Leucker, D. Neider, A survey of model learning techniques for recurrent neural networks, Springer, 2022, pp. 81–97.

[5] R. Johnson, T. Zhang, Supervised and semi-supervised text categorization using lstm for region embeddings, in: International Conference on Machine Learning, PMLR, 2016, pp. 526–534.

[6] D. Angluin, Learning regular sets from queries and counterexamples, Information and computation 75 (1987) 87–106.

[7] F. Mayr, S. Yovine, R. Visca, Property checking with interpretable error characterization for recurrent neural networks, Machine Learning and Knowledge Extraction 3 (2021) 205–227.

[8] E. Muškardin, B. K. Aichernig, I. Pill, M. Tappler, Learning finite state models from recurrent neural networks, in: International Conference on Integrated Formal Methods, Springer, 2022, pp. 229–248.

[9] E. Muškardin, M. Tappler, I. Pill, B. K. Aichernig, T. Pock, On the relationship between rnn hidden state vectors and semantic ground truth, 2023.

[10] D. Hong, A. M. Segre, T. Wang, Adaax: Explaining recurrent neural networks by learning automata with adaptive states (2022) 574–584.

[11] Q. Wang, K. Zhang, A. G. Ororbia II, X. Xing, X. Liu, C. L. Giles, A comparative study of rule extraction for recurrent neural networks, arXiv preprint arXiv:1801.05420 (2018).

[12] Z. Wei, X. Zhang, Y. Zhang, M. Sun, Weighted automata extraction and explanation of recurrent neural networks for natural language tasks, Journal of Logical and Algebraic Methods in Programming 136 (2024) 100907.

[13] D. Song, X. Xie, J. Song, D. Zhu, Y. Huang, F. Juefei-Xu, L. Ma, Luna: A model-based universal analysis framework for large language models, IEEE Transactions on Software Engineering 50 (2024) 1921–1948.

[14] G. Dong, J. Wang, J. Sun, Y. Zhang, X. Wang, T. Dai, J. S. Dong, X. Wang, Towards interpreting recurrent neural networks through probabilistic abstraction (2020) 499–510.

[15] W. Merrill, N. Tsilivis, Extracting finite automata from rnns using state merging, 2022.

[16] E. M. Gold, Language identification in the limit, Information and control 10 (1967) 447–474.

[17] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977, pp. 238–252.

[18] T. Le Gall, B. Jeannet, Lattice automata: A representation for languages on infinite alphabets, and some applications to verification, in: International Static Analysis Symposium, Springer, 2007, pp. 52–68.

[19] A. Lemesle, J. Lehmann, T. L. Gall, Neural network verification with pyrat, arXiv preprint arXiv:2410.23903 (2024).

[20] A. Soubki, J. Heinz, Benchmarking state-merging algorithms for learning regular languages, in: International Conference on Grammatical Inference, PMLR, 2023, pp. 181–198.