DeepDFA: Automata Learning Through Neural Probabilistic Relaxations

Elena Umili[a,*] **and Roberto Capobianco**[b]

[a]Sapienza University of Rome
[b]Sony AI

Abstract. In this work, we introduce DeepDFA, a novel approach to identifying Deterministic Finite Automata (DFAs) from traces, harnessing a differentiable yet discrete model. Inspired by both the probabilistic relaxation of DFAs and Recurrent Neural Networks (RNNs), our model offers interpretability post-training, alongside reduced complexity and enhanced training efficiency compared to traditional RNNs. Moreover, by leveraging gradient-based optimization, our method surpasses combinatorial approaches in both scalability and noise resilience. Validation experiments conducted on target regular languages of varying size and complexity demonstrate that our approach is accurate, fast, and robust to noise in both the input symbols and the output labels of training data, integrating the strengths of both logical grammar induction and deep learning.

1 Introduction

The problem of identifying a deterministic finite state automaton (DFA) from labeled traces is one of the best-studied problems in grammatical inference [11]. The latter sees applications in various areas, including Business Process Management (BPM) [1], non-Markovian Reinforcement Learning [14, 26], automatic control [7], speech recognition [3], and computational biology [25]. Both passive [17] and active [4] exact methods have been proposed for DFA identification. These methods are guaranteed to succeed in theory. However, in practice, they require a notable amount of computation, and they are unable to handle errors in the training dataset, making them of limited applicability, especially to real applications and large DFAs. Unlike exact approaches, Recurrent Neural Networks (RNN) tolerate errors in the training data, and they have proven highly effective at learning classifiers for sequential data [12]. DFAs and RNNs can both be used for language recognition, which is essentially binary classification over strings. Many works highlight the similarities between RNNs and finite-state machines [27]. The two differ in the transition representation: RNNs learn a parametrized transition function in a continuous hidden state space, while DFAs have a finite state space and completely transparent transitions. Furthermore, designing an RNN requires many choices: deciding the number of layers, the number of features of each hidden layer, and all the activation functions. Each of these decisions can affect the final performance and must be taken carefully. By contrast, exact methods do not require nearly any hyperparameter fine-tuning. Many approaches have been proposed to extract a DFA from a pre-trained RNN [24, 32, 33], generally adapting techniques from the grammar induction literature and

suggesting ways to discretize or cluster the continuous RNN hidden states in a finite structure. The purpose of these works is not DFA induction but rather to enhance the explainability of black-box sequence classifiers. However, they open the door to DFA induction through neural networks, and join two fields that are classically kept separated.

In order to take the benefits from both worlds, grammar induction on one side and recurrent neural networks on the other side, we present DeepDFA: a transparent neural network design specialized in learning DFAs from traces with gradient-based optimization. The model resembles a recurrent neural network, but, differently from RNNs, it is completely transparent after training, as much as a DFA. Furthermore, thanks to its specialized design, it uses fewer weights than the most commonly used recurrent architectures, such as LSTMs and GRUs, and it only has one hyperparameter. This results in faster training and less memory consumption and a significantly reduced hyperparameter search. At the same time, since it is trained with back-propagation, it is able to learn DFAs in a significantly shorter time than grammar induction methods, even for large automata; and it can tolerate errors in the training labels and the training symbols. Our method is based on defining a neural network that behaves as a probabilistic finite automaton. We control how much the probabilities are close to categorical one-hot distribution through a temperature value. During training, we smoothly drive the network activations to be close to discrete 0/1 values by changing the temperature. When the gap between the discrete and actual activations is small enough, the network behaves precisely as a DFA.

We evaluate our method on the popular Tomita languages benchmark [28] and random DFAs of different sizes and different sets of symbols. Results show that DeepDFA is fast and accurate. It outperforms an exact SAT-based method [35] when the target DFA has more than 20 states, or the training set contains mislabeled examples, and it reaches competitive results in the other cases. We also compared DeepDFA to DFA extraction from a pre-trained RNN [33], finding it reaches better accuracy and predicts DFA of size closer to the target DFA size. The remainder of this paper is organized as follows: in section 2 we report related works; in section 3 we give some preliminaries on Deterministic and Probabilistic Finite Automata and Recurrent Neural Networks; in section 4 we formulate our problem and illustrate in detail the framework used to solve it; we report the experiments evaluating our approach in section 5; and finally we conclude and discuss directions for future work in section 6.

---

* Corresponding Author. Email: umili@diag.uniroma1.it

2   Related works

Combinatorial methods for grammar induction   Many approaches have been proposed to identify a target DFA from a set of positive and negative traces. The L* algorithm [4], is an exact active learning algorithm to learn a DFA from an oracle through membership and equivalence queries. Another approach is to apply the evidence-driven state-merging algorithm [22, 8], which is a greedy algorithm that is not guaranteed to converge to the global optimum. A more modern approach is to leverage highly-optimized SAT solvers [17]. This approach is guaranteed to find the minimal DFA consistent with all the training examples, but suffers from scalability problems. In this regard, several symmetry-breaking predicates have been proposed for the SAT encoding to reduce the search space [34, 35].

DFA extraction from recurrent neural networks   Prior works extract a DFA from a pre-trained RNN, to explain the network behavior. Weiss et al. [33] adapt the L* algorithm [4] to work with an RNN oracle. Other work uses k-means clustering on the RNN hidden states to extract a graph of states [32]. Merrill et al. extract a DFA from an RNN by first building a prefix tree and then merging states with close state representation in the RNN hidden space [24]. These approaches train an RNN with the labeled strings and then extract an equivalent DFA from the trained model. Our approach differs from these since we directly train an RNN equivalent to a Probabilistic Finite Automaton (PFA), and we force the probabilities to become close to one-hot categorical distributions *during training*. In this way, our model *becomes a DFA*. In other words, there is no difference between the trained neural model and the automaton, and there is no risk that the abstraction does not represent the network, as for previous works. For example, Wang et al. [32] cluster the RNN states, so the automaton depends on the clustering algorithm performance. Performances from [24] instead rely on a similarity threshold and, as the paper shows, also on the number of epochs the RNN is trained after convergence. Our work in this sense is more similar to [33], since it computes the abstraction *automatically*. The difference is in how the abstraction is computed: we use gradient descent optimization while [33] starts with a hand-crafted discretization that is automatically refined during automata learning with L*.

Learning crispy logical models through gradient descent   Classically, the induction of logic models, including DFAs, is not approached with gradient-descent optimization methods (as deep learning methods) since their finite and 'crispy' nature prevents the gradient computation. However, recent works in neurosymbolic AI [15] propose techniques to reduce the gap between the induction of crispy models and that of continuous ones. Walke et al. [31] proposes a recurrent neural model with specialized filters to learn Linear Temporal Logic formulas over finite traces (LTLf) from labeled traces. Collery et al. [9] discovers logical rules describing patterns in sequential data using a differentiable model. Aichernig et al. [2] learn the DFA by constraining the outputs of a simple RNN to be close to one-hot vectors through a regularization term in the loss. Differently from [2], we define a new recurrent model specialized in learning PFAs, which we discretize at training time with temperature annealing and train with the classical cross-entropy between the predicted and the ground-truth label. Grachev et al. [16] proposes a neural network model similar to ours to learn DFA from traces. However, this model does not use activation functions with discrete outputs, and it is affected by the vanishing gradient problem for automata larger than 5 states, while our method can effectively learn target automata with up to 30 states.

3   Background

3.1   Deterministic Finite Automata

A Deterministic Finite Automaton (DFA) A is a tuple (P, Q, δ, q_0, F), where P is a set of propositional symbols called the alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times P \rightarrow Q$ is the transition function, and $F \subseteq Q$ is the set of final states. Let P^* be the set of all finite strings over P, and ϵ the empty string. The transition functions over strings $\delta^* : Q \times P^* \rightarrow Q$ is defined recursively as

$$\begin{aligned} \delta^*(q, \epsilon) &= q \\ \delta^*(q, ax) &= \delta^*(\delta(q, a), x) \end{aligned} \tag{1}$$

Where $a \in P$ is a symbol and $x \in P^*$ is a string, and ax is the concatenation of a and x. A accepts the string x (i.e., x is in the language of A, $L(A)$) if and only if $\delta^*(q_0, x) \in F$. Let $x = x[1]x[2]...x[l]$ be the input string, where $x[t]$ is the t-th character in the string, we denote as $q = q[0]q[1]...q[l]$ the sequence of states visited by the automaton while processing the string, namely $q[0] = q_0$ and $q[t] = \delta(q[t-1], x[t])$ for all $t > 0$.

3.2   Probabilistic Finite Automata

A Probabilistic Finite Automaton (PFA) A_p is a tuple $(P, Q, i_p, \delta_p, f_p)$, where P is the alphabet, Q is the finite set of states, $i_p : Q \rightarrow [0, 1]$ is the probability for a state to be an initial state, $\delta_p : Q \times P \times Q \rightarrow [0, 1]$ is the transition probability function, and $f_p : Q \rightarrow [0, 1]$ is the probability of a state to be final. We have therefore $\sum_{q' \in Q} \delta_p(q, p, q') = 1$, and $\sum_{q \in Q} i(q) = 1 \; \forall q \in Q, \forall a \in P$.

We can represent the PFA in matrix form as a *transition matrix* T, an *input vector* v_i and an *output vector* v_o. Matrix $T \in \mathbb{R}^{|P| \times |Q| \times |Q|}$ contains at index (p, q, q') the value of $\delta_p(q, p, q')$. We denote as $T[p] \in \mathbb{R}^{|Q| \times |Q|}$ the 2D transition matrix for symbol p.

The input vector $v_i \in \mathbb{R}^{1 \times |Q|}$ contains at index k the probability of state q_k to be an initial state, while the output vector $v_o \in \mathbb{R}^{|Q| \times 1}$ has in position k the probability of state q_k to be accepting. This matrix representation is shown in Figure 1(b).

Given a string $x = x[0]x[1]...x[l-1]$, we denote as $q_{p,0}, q_{p,1}...q_{p,l}$ the sequence of probabilities to visit a certain state, where $q_{p,t} \in \mathbb{R}^{1 \times |Q|}$ is a row vector containing at position k the probability to stay in state k at time t.

$$\begin{aligned} q_{p,0} &= v_i \\ q_{p,t} &= q_{p,t-1}T[x[t]] \qquad \forall t > 0 \end{aligned} \tag{2}$$

The probability of being in a final state at time t is the inner product $q_{p,t}v_o$.

Therefore the probability of a string to be accepted is the probability to be in a final state in the last computed state $q_{p,l}$, and it is calculated as follows

$$v_i T[x[0]]T[x[1]]..T[x[l-1]]v_o \tag{3}$$

Figure 1 shows a comparison between a DFA and a PFA.

3.3   Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a parameterized function $h_t = f(h_{t-1}, x_t; \theta_h)$, having trainable parameters θ_h, that takes as input a state-vector at time $t - 1$, h_{t-1}, and the input vector at time t, x_t, and returns the current state-vector at time t, h_t. An RNN
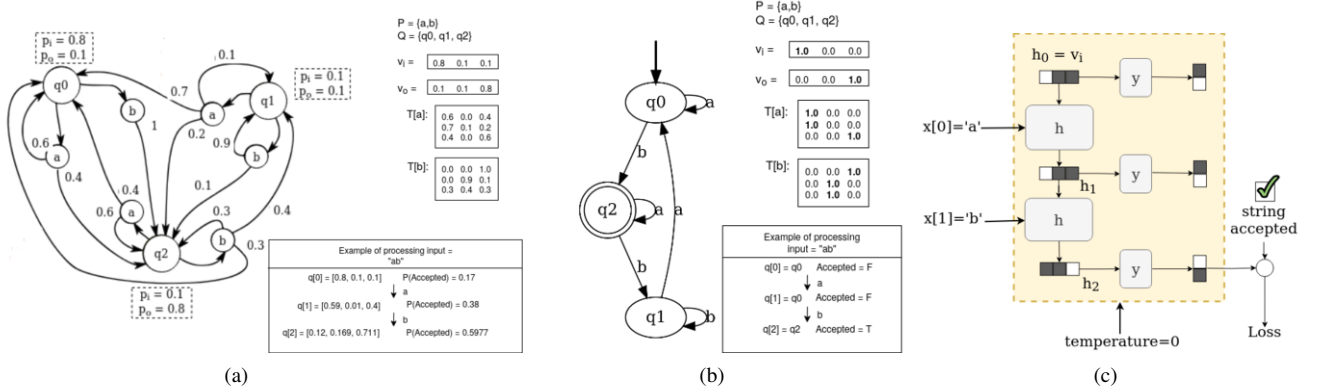
Figure 1: a) An example of PFA with three states and two symbols: graph describing the PFA, equivalent representation in matrix form, and produced states and acceptance probabilities while processing the string "ab". b) An example of DFA: graph describing the PFA, equivalent representation in matrix form, and produced states and acceptance probabilities while processing the string "ab". In particular, the DFA in (b) is obtained by the PFA in (a) approximating the matrix representation to the closest one-hot vectors. c) DeepDFA processing the string "ab".

can be applied to a sequence $x[0], ..., x[n]$ by recursive application of the function h to the vectors $x[i]$. An example of RNNs are the Elman and Jordan networks [13], also known as Simple Recurrent Networks.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y) \tag{4}$$

Where x is the network input, h is the hidden state, y is the network output, W_h, U_h, W_y and b_h are the network parameters, and σ_h and σ_y are activation functions.

RNNs are employed in a variety of tasks on sequential data, including sequence classification. A binary RNN-acceptor, is an RNN where y classifies the RNN's state vectors as accepting or rejecting. An RNN-acceptor is conceptually equal to a DFA, except that it processes a sequence by applying continuous state-transitions and output evaluations, that are usually hard to inspect and explain. In our work we propose an alternative RNN model tailored on DFAs that makes the model interpretable by design.

4 DeepDFA

We consider the problem of inferring a DFA from a training set of labeled strings $D = \{(x_1, \bar{y}_1), (x_2, \bar{y}_2), ..., (x_n, \bar{y}_n)\}$, where x_i is a string of length l of symbols $x[0], x[1], ..., x[l-1]$, in the automaton alphabet P, $x_i[t] \in P$ with $0 \leq t < l$, and $\bar{y}_i \in \{0, 1\}$ is the associated label, denoting whether the string is accepted or not by the target automaton.

To infer the automaton, we define a recurrent neural network model that mimics the behavior of a PFA in a state space $\hat{Q}_{max}$ and action space P, where P is the target automaton alphabet while $\hat{Q}_{max}$ is our hypothesis on the state space, which will generally differ from the true Q. For this reason, the size of $\hat{Q}_{max}$ is a hyperparameter of our model.

We cannot use gradient-based optimization to learn the DFA directly because of its non-differentiable transitions and output vector. The intuition behind our work is that PFAs are closely related to recurrent neural networks, since they calculate the next state and output using multiplications between continuous vectors and matrices, in the same way RNNs do. However, DFAs can also be represented in matrix form, with the difference that their matrix representation is composed of only one-hot row vectors. As example we show in Figure 1 a simple PFA (subfigure (a)) and the DFA obtained by approximating all the PFA matrix representation row vectors to the closest one-hot (subfigure (b)).

Following this idea, we define DeepDFA as a parametric PFA in which we can drive the representation to be close to one-hot during training. We obtain this effect using an activation function that smoothly approximates discrete output. Many works use this technique [20, 31, 21], especially in neurosymbolic AI [5], to learn symbolic logic structures by using differentiable models such as neural networks. In particular, we use a modified version of the classical softmax and sigmoid activation functions, which we call $softmax\_with\_temp$ and $sigmoid\_with\_temp$. Given a function $f(x)$ we define $f\_with\_temp(x, \tau) = f(x/\tau)$, with τ being a positive temperature value.

Our RNN comprises two trainable modules: a transition function and an output function. The transition function $h_t(h_{t-1}, x[t-1]; \theta_h)$ has parameters θ_h, takes as input the probabilities over the previous state, $h_{t-1} \in [0, 1]^{|\hat{Q}|max}$, and the previous symbol, $x[t-1] \in P$, and predicts the current state h_t. The output function $y(h_t; \theta_y)$ implements the classification module: given the current state estimation, h_t, predicts the probability of the current state to be an accepting state using its parameters θ_y. In particular, fixed a temperature value τ

$$\begin{aligned} h_0 &= v_i = [1, 0, ..., 0] \\ h_t &= h_{t-1} T[x[t-1]] \\ y_t &= h_t v_o \\ T &= softmax\_with\_temp(\theta_h, \tau) \\ v_o &= sigmoid\_with\_temp(\theta_y, \tau) \end{aligned} \tag{5}$$

Where T is the PFA transition matrix, $T[x[t]]$ is the transition matrix for symbol $x[t]$, and v_o is the output vector, as defined in Section 3.2. They are obtained by applying discrete activation functions on parameters $\theta_h \in \mathbb{R}^{|P| \times |\hat{Q}_{max}| \times |\hat{Q}_{max}|}$ and $\theta_y \in \mathbb{R}^{|\hat{Q}_{max}| \times 1}$. In particular, the softmax activation applied to the third dimension of the matrix θ_h ensures $\sum_{q'} \delta_p(q, a, q') = 1$, and the sigmoid activation ensures values of v_o stay in $[0, 1]$.

In the forward pass, we calculate the probability of a string x in the dataset to be accepted by the RNN by applying Equation 5 recursively to the input sequence of symbols $x[0], x[1], ..., x[l-1]$. The final output y_l is compared with the ground truth label $\bar{y}$. We update the model weights with back-propagation by minimizing the binary cross-entropy between model predictions and the ground truth labels.

Temperature Annealing Cold temperatures force the PFA to behave as a DFA, since the activation values become closer to boolean values as the temperature decreases. When the temperature is low enough, the model transforms into a DFA. Let us notice that using

the classical activation functions and discretizing the model at the end is not guaranteed to work because the discretized model differs from the trained one, and the two can have different performances. However, using a cold temperature from the start of training can prevent the system from learning the model properly. For this reason, we initialize τ to 1. In this way, the system starts the training using the normal softmax and activation functions. After that, we anneal the temperature to 0 by multiplying it by a positive constant $\lambda < 1$ at each epoch. In Figure 1(c), we show the network behavior in case of a perfectly discretized activation, namely when the temperature is 0. In theory, this ideal behavior is obtained only with zero temperature, but in practice, we observe the continuous and discretized models start having the same accuracy already at temperatures around 0.8.

Model Minimization Once the training is concluded, we read the DFA from the activations evaluated with the minimum temperature, and we use the Hopcroft algorithm [19] to minimize the number of states. We emphasize that automata minimization is a well-known problem for which many algorithms are available, as opposed to neural network compression, also known as knowledge distillation [18], which is still an open research problem. This represents another feature in which DeepDFA can take 'the best of both worlds.' We observe that, even if the state space size $|\hat{Q}_{max}|$ is large, the number of states after minimization $|\hat{Q}|$ tends to be close to the target DFA number of states, which suggests the model is robust to overfitting.

Extension to Probabilistic Symbols DeepDFA allows us to represent uncertainty over both the transition and the output function, because it is based on representing the model as a probabilistic machine. However, let us notice the current symbol $x[i]$ is used to index the transition matrix in Equations 5 and, as a consequence, *symbols must be integers*. This contrasts with symbol grounding techniques based on neural networks, which usually predict a *probabilistic belief* on symbol truth values. For this reason, we extend the framework to be fully probabilistic, and consider probability values over symbols in the calculations. Given a sequence of inputs $x_1, x_2, ..., x_l$, where x_i is a probability vector over $|P|$ classes, we define the next state and output of the network as follows

$$
\begin{aligned}
h_0 &= v_i = [1, 0, ..., 0] \\
h_t &= \sum_{p=0}^{p=|P|} x_{t-1,p}(h_{t-1}T[p]) \\
y_t &= h_t v_o \\
T &= softmax\_with\_temp(\theta_h, \tau) \\
v_o &= sigmoid\_with\_temp(\theta_y, \tau)
\end{aligned} \tag{6}
$$

where we denote as $x_{t,p}$ the probability that x_t is symbol $p \in P$.

5   Experimental Evaluation

In this section, we report the main results supporting our method. We provide our code at https://github.com/whitemech/DeepDFA.

Target DFAs We test our approach on two different sets of DFAs. The first experiment is on the Tomita languages [28], which are a standard benchmark for grammar induction and DFA extraction from RNNs [33, 32]. The benchmark comprises seven formal languages of increasing difficulty defined on the binary alphabet $P = \{a, b\}$. Despite Tomita languages being a popular benchmark, they are represented by small DFAs with state size smaller than six. In order to test our approach on bigger DFAs, we conduct a second experiment on randomly generated DFAs with state size $|Q|$ between 10 and 30, and alphabet size $|P|$ between 2 and 3. For each setting, we generate

5 random DFAs as described in [34]. $|Q|$ states are generated and enumerated between 1 and $|Q|$, we set 1 as the initial state, and each state is equiprobable to be accepting. We connect each state i with a random state in $[i + 1, Q]$ with a random-labeled transition. In this way, we partially build an automaton where all states are reachable from the initial state. Finally, we complete the DFA with random transitions.

Dataset For each target DFA, we create a train, a dev, and a test dataset by sampling strings of various lengths and labeling them with the target DFA. The training dataset contains strings with a length between 1 and len_{train}, the dev set is composed of strings of length len_{dev}, and the test set by sequences of length len_{test}. In order to test the model's capability to generalize to longer unseen sequences, for each dataset, we set $len_{train} < len_{dev} < len_{test}$. To prevent the models from learning degenerate solutions, all the train datasets are nearly perfectly balanced. We set $len_{train} = 30$, $len_{dev}=60$, and $len_{test}=90$ for all the Tomita datasets and the random DFAs datasets with $|Q| < 30$. For the random DFAs with state size of 30, we set $len_{train}=50$, $len_{dev}=100$, and $len_{test}=150$. We report the size of each dataset in [29]. To test the resiliency of different methods to errors in the training data, we also create a corrupted version of each training dataset by flipping 1% of the labels.

Baselines Our approach is a hybrid between a RNN and a DFA. These two types of sequence acceptors are trained with very different methods and present different strengths and weaknesses. In order to better understand the benefits of having a hybrid method, we compare it with one state-of-the-art from the literature on grammar induction, **DFA inductor** [35], one state-of-the-art method to extract DFAs from RNNs, **L* extraction** [33], and one state-of-the-art neurosymbolic method for automata learning, **DFA Generator** [16]. Notice we cannot compare with the other neurosymbolic methods cited in Section 2. Indeed, Walke et al. [31] learns an LTLf formula from data, LTLf formulas are less expressive than DFAs, since they can capture only *star-free* regular languages [10], while DFAs can capture both star-free and non-star-free languages. Note that Tomita 3,5 and 6 are non-star-free grammars [6], and our schedule to produce random DFAs may generate non-star free regular languages. Collery et al. [9] learn a set of local and global rules that are ultimately combined, and the relationship between their formalism and that of DFAs is not clear. Regarding the chosen baselines, we remind that DFA-inductor is a SAT-based approach for exact DFA identification. In particular, we use the shared implementation code at [1], and we test with breadth-first search (BFS) symmetry breaking, shown to be the most effective in the paper. Instead, L* extraction abstracts a finite state automaton from a pretrained RNN, starting from a predefined discretization of the hidden state space and applying the L* algorithm and abstraction refinement when required. To test this method, for each language: (i) we train many RNNs of different types, different number of layers and different hidden-state size, (ii) we record the performances on the dev set (iii) we apply L* extraction on the RNN achieving the best dev accuracy. In particular, we use the code in the public repository [2] with 10 as the initial split depth, and the starting examples set composed of the shortest positive string and the shortest negative string in the train set, as suggested in the paper. DFA Generator trains directly a neural network shaped as a DFA, as our method. The algorithm needs as hyperparameter the maximum number of states $\hat{Q_{max}}$. We train with $\hat{Q_{max}}$ of different sizes (the

---

Table 1: Comparison between **DeepDFA**, **L* extraction**, **DFA-inductor** and **DFA-generator** on the **Tomita** Languages. We report test accuracy, mean number of states $|\hat{Q}|$, and the number of parameters used $\#W$.

Lang	DeepDFA			L* extraction			DFA-inductor		DFA-generator		
	Accuracy	$|\hat{Q}|$	$\#W$	Accuracy	$|\hat{Q}|$	$\#W$	Accuracy	$|\hat{Q}|$	Accuracy	$|\hat{Q}|$	$\#W$
T1 ($|Q|=2$)	**100 ± 0**	**2 ± 0**	220	**100 ± 0**	**2 ± 0**	3030	**100 ± 0**	**2**	100 ± 0	1.8 ± 1.1	**210**
T2 ($|Q|=3$)	**100 ± 0**	**3 ± 0**	220	**100 ± 0**	**3 ± 0**	3030	**100 ± 0**	**3**	100 ± 0	1 ± 0	**210**
T3 ($|Q|=5$)	**100 ± 0**	**5 ± 0**	220	**100 ± 0**	**5 ± 0**	3030	**100 ± 0**	**5**	91 ± 15	5 ± 1.6	**210**
T4 ($|Q|=4$)	**100 ± 0**	**4 ± 0**	220	**100 ± 0**	**4 ± 0**	3030	**100 ± 0**	**4**	85 ± 33	4.8 ± 0.4	**210**
T5 ($|Q|=4$)	**100 ± 0**	**4 ± 0**	220	**100 ± 0**	**4 ± 0**	62000	**100 ± 0**	**4**	100 ± 0	2.6 ± 0.9	**210**
T6 ($|Q|=3$)	**100 ± 0**	**3 ± 0**	220	91.1 ± 19.8	170.6 ± 374.7	41400	**100 ± 0**	**3**	100 ± 0	3 ± 0	**210**
T7 ($|Q|=5$)	**100 ± 0**	**5 ± 0**	220	**100 ± 0**	**5 ± 0**	3030	**100 ± 0**	**5**	100 ± 0	4.4 ± 0.9	**210**

Table 2: Comparison between **DeepDFA**, **L* extraction**,**DFA-inductor** and **DFA-generator** on the **noisy dataset** constructed for the **Tomita** Languages. We report test accuracy, mean number of states $|\hat{Q}|$, and the number of parameters used $\#W$.

Lang	DeepDFA			L* extraction			DFA-inductor		DFA-generator		
	Accuracy	$|\hat{Q}|$	$\#W$	Accuracy	$|\hat{Q}|$	$\#W$	Accuracy	$|\hat{Q}|$	Accuracy	$|\hat{Q}|$	$\#W$
T1 ($|Q|=2$)	**100 ± 0**	**2 ± 0**	1860	**100 ± 0**	**2 ± 0**	3030	95	12	100 ± 0	2.2 ± 1.1	**210**
T2 ($|Q|=3$)	**100 ± 0**	**3 ± 0**	220	**100 ± 0**	**3 ± 0**	4020	94	10	100 ± 0	1 ± 0	**210**
T3 ($|Q|=5$)	**100 ± 0**	**5 ± 0**	220	100 ± 0	7.4 ± 3.3	4020	o.o.m.	–	100 ± 0	4.8 ± 0.8	**210**
T4 ($|Q|=4$)	**100 ± 0**	**4 ± 0**	**220**	**100 ± 0**	**4 ± 0**	4020	o.o.m.	–	90 ± 22.3	5.8 ± 1.3	1830
T5 ($|Q|=4$)	**100 ± 0**	**4 ± 0**	220	**100 ± 0**	**4 ± 0**	62000	o.o.m.	–	100 ± 0	2.6 ± 0.8	**210**
T6 ($|Q|=3$)	**100 ± 0**	**3 ± 0**	220	99.9 ± 0.1	107 ± 170.6	41400	o.o.m.	–	**100 ± 0**	**3 ± 0**	**210**
T7 ($|Q|=5$)	**100 ± 0**	**5 ± 0**	220	**100 ± 0**	**5 ± 0**	31100	o.o.m.	–	100 ± 0	4.4 ± 1.1	**210**

Table 3: Comparison between **DeepDFA**, **L* extraction**, **DFA inductor** and **DFA-generator** on **random DFAs**. We keep the best 5 experiments over 10 for for the stochastic methods.

$|Q|$	$|P|$	DeepDFA			L* extraction			DFA-inductor			DFA-generator		
		Test Acc.	$|\hat{Q}|$	Exec. Time	Test Acc.	$|\hat{Q}|$	Exec. Time	Test Acc.	$|\hat{Q}|$	Exec. Time	Test Acc.	$|\hat{Q}|$	Exec. Time
10	2	**100 ± 0**	**8.0 ± 0**	52 ± 13	99.9 ± 0.1	16.2 ± 41.2	124 ± 21	100 ± 0	7.8 ± 0.44	**42 ± 5**	87 ± 19.7	16 ± 17.5	2867.7 ± 230.5
10	3	100 ± 0	10.6 ± 2.2	**43 ± 15**	97.4 ± 3.7	296.1 ± 223.3	208 ± 42	**100 ± 0**	**10 ± 0**	129 ± 2	o.o.t.	–	–
20	2	**100 ± 0**	**14.0 ± 1.8**	49 ± 14	99.9 ± 0.2	196.7 ± 281.8	167 ± 37	100 ± 0	14 ± 2	245 ± 117	o.o.t.	–	–
20	3	99.9 ± 0.2	18.7 ± 1.7	**55 ± 12**	69.5 ± 4.8	389.6 ± 203.4	334 ± 52	**100 ± 0**	**18.4 ± 1.1**	857 ± 238	o.o.t.	–	–
30	2	**100 ± 0**	**22.6 ± 1.3**	988 ± 525	99.9 ± 0	77.0 ± 124.9	1216 ± 122	o.o.m.	–	–	o.o.t	–	–
30	3	**100 ± 0**	**28.0 ± 1.0**	866 ± 447	61.7 ± 13.8	372.9 ± 171.0	2424 ± 684	o.o.m.	–	–	o.o.t.	–	–

Table 4: Comparison between **DeepDFA**, **L* extraction**, **DFA inductor** and **DFA-generator** on the **noisy dataset** created for the **random DFAs**.

| $|Q|$	$|P|$	DeepDFA			L* extraction			DFA-inductor		DFA-generator		
		Test Acc.	$|\hat{Q}|$	$\#W$	Test Acc.	$|\hat{Q}|$	$\#W$	Test Acc.	$|\hat{Q}|$	Test Acc.	$|\hat{Q}|$	$\#W$
10	2	93.6 ± 10.7	**11.5 ± 5.25**	20200	**98.4 ± 5.0**	131.5 ± 225.4	62000	o.o.m.	–	84 ± 14.5	15 ± 18	**5050**
10	3	**95.1 ± 3.6**	**37.2 ± 11.1**	30200	92.0 ± 8.4	290.5 ± 193.2	82600	o.o.m.	–	o.o.t.	–	–
20	2	**99.6 ± 1.7**	**14.7 ± 3.2**	20200	98.9 ± 1.9	359.0 ± 251.2	62000	o.o.m.	–	o.o.t.	–	–
20	3	**96.0 ± 4.1**	**34.9 ± 10.1**	30200	66.2 ± 4.5	420.3 ± 220.3	82600	o.o.m.	–	o.o.t.	–	–
30	2	**99.6 ± 1.5**	**22.6 ± 1.7**	80400	98.2 ± 7.4	238.9 ± 252.2	122200	o.o.m.	–	o.o.t.	–	–
30	3	**98.7 ± 2.0**	**55.1 ± 34.0**	120400	60.4 ± 11.3	338.7 ± 173.4	326800	o.o.m.	–	o.o.t.	–	–

same used for our method when it is possible), and we use the implementation at the public repository [3]. For each approach, we report the accuracy obtained by querying the final DFA on the test set, the number of states of the predicted DFA $|\hat{Q}|$, the execution time (in seconds), and, when applicable, the number of weights of the model.

Training Details We train all the neural networks on an Nvidia GPU GeForce GTX 1650 Ti, with a learning rate of 0.01 for DeepDFA and of 0.001 for the RNNs, until training loss convergence, for a maximum of 200 epochs. In all the experiments we use $\lambda = 0.999$ and minimum temperature $= 10^{-5}$. All the experiments of DFA-inductor were performed on a Intel Core i7-10750H CPU, without any other process running at the same time.

5.1 Results

Results on Tomita Languages For each Tomita language, we report the results averaged over 5 experiments with different random seeds for L* extraction, DFA-generator and DeepDFA, and the results of one application of DFA-inductor, since the latter has a deterministic behavior. To explore the influence of hyperparameter choices on DeepDFA, DFA-generator and L* extraction, we test 3 different architectures for DeepDFA and DFA-generator and 6 different RNNs for L*. In particular, we test DeepDFA and DFA-generator with the maximum number of states equal to 10, 30, and 100 (denoting the models as DeepDFA($|\hat{Q}_{max}|$) and DFAGen(($|\hat{Q}_{max}|$)). We test L* extraction on both LSTMs and GRUs, for each type of RNN we test three architectures (denoted here as RNN(#layers, #hidden neurons per layer)), namely: RNN(1,30), RNN(1,100), RNN(2,100). For space reasons, we report in [29] the results obtained for each model tested and here only the results of the best model for each approach. We select he best model per approach as the more accurate on
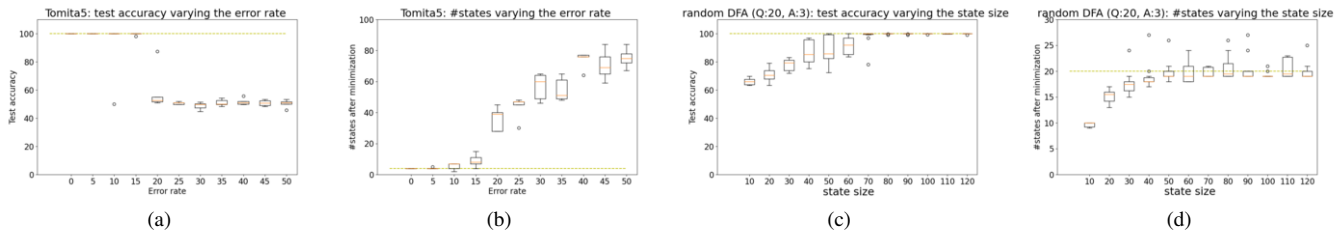
Figure 2: Ablation study 1: **results on Tomita 5 with different error rates in the training dataset**. a) Test accuracy. b) Number of states of predicted DFAs. Ablation study 2: **results a random DFA of size 20 and alphabet size 3 varying the state size hyperparameter**. c) Test accuracy. d) Predicted number of states. Each box represents 10 experiments performed with different random seeds. The box extends from the lower to upper quartile values of the data, with the line at the median. The whiskers extend from the box to show the range of the data. Flier points represent outliers.

the development set. In cases where two or more models achieve the same highest accuracy on the development set, we prioritize selecting the model with the smallest number of weights among them. Results for the Tomita benchmark are reported in Table 1. For space reasons we do not report the execution time for the Tomita languages. However each run last less than a minute across the all 7 languages for all the models tested for all the approaches, except for DFA-generator. This proved to be the slowest of all the methods tested. In particular we set a threshold time of one hour for a run and we denote with 'o.o.t' (out of time) when a method exceed the threshold time. DFA-Gen(10) takes less than a minute only for Tomita 1, 2 and 3, and never exceed the timeout. DFAGen(30) execution time is less than a minute only for Tomita 1 and 2 and exceed the maximum time for Tomita 5 and 7, while DFAGen(100) always goes out of maximum time. Remarkably, both our approach and DFA-inductor achieve perfect performances on all the the 7 languages, namely a test accuracy of 100% and a predicted number of states equal to the target. The same is achieved by L* extraction, for all the grammars except Tomita 6, for which it predicts extremely oversized DFAs with a mean of 170 states. DFA-generator instead can produce optimal results only for Tomita 6 and has lower performances on all the other languages. To evaluate how noise in the training data affects the performance of the four methods, we replicate the experiment using training datasets with 1% label noise. The results are presented in Table 2. Results show that DFA-inductor is completely unable to handle errors in the training data. In fact, with only 1% of errors, the process is killed for exceeding the CPU capacity before finding a DFA consistent with training examples. In the table, these instances are marked with an 'o.o.m' (out of memory). The method crashes in all the Tomita languages except the first two, for which it loses around 5% of accuracy and overestimates the state space. As expected, the other methods based on neural networks exhibits greater robustness to training set errors. DFA-generator presents non optimal performances similar to the case without noise. L* extraction overestimates the number of states for Tomita3 and 6. In contrast, our approach remains unaffected by minor errors in the training dataset and consistently maintains the top performance levels achieved with the error-free dataset. As an ablation study, we subjected DeepDFA to different error rates (ranging from 0 to 50%) on Tomita 5. Each noise configuration was tested across five experiments, with $|\hat{Q}_{max}|$ set to 200. The results in Figure 2(a-b) show that our model can tolerate larger error rates of up to 15% achieving 100% test accuracy and a state count similar to the target one.

Results on randomly generated DFAs We generate random DFAs with varying state and alphabet sizes using the approach outlined in the 'Target DFAs' paragraph. For each randomly generated DFA, we conduct 10 tests for each stochastic methods (DeepDFA, L* extrac-

tion and DFA-generator) and one test with DFA-inductor. For this benchmark we set a larger threshold time of two hours for each run. We configure DeepDFA with a state size $|\hat{Q}max|$ of 100 for random DFAs with state sizes of 10 and 20, and $|\hat{Q}max|$ of 200 when the desired number of states is 30. When testing L* extraction, we train both LSTM and GRU RNNs of one and two layers, using the same hidden state sizes used for DeepDFA. We initially tested DFA-generator with $|\hat{Q}max|$ set to 100, only to find that it always exceeded the time limit. Consequently, we experimented with a reduced $|\hat{Q}max|$ value of 50. Despite this adjustment, the method continued to go out-of-time for the majority of languages. This happens for a difficulty in reducing the loss function, causing the method to remain stalled. However, this confirms the scalability challenges previously acknowledged by the authors of the method in [16], highlighting the method's struggle to converge when dealing with automata larger than five states. Our comparison of the four methods is presented in Table 3. For the stochastic methods, we provide the average performance over the best 5 out of 10 runs. DFA-inductor is able to reach top test accuracy unless it crashes due to exceeding CPU capacity; this again happens for the biggest-size DFAs of size 30. Our approach competes favorably with DFA-inductor for target DFAs smaller than size 30, and it excels in cases where DFA-inductor falls short. Compared to L* extraction and DFA-generator, our method consistently exhibits higher accuracy. Notably, L* extraction tends to significantly overestimate the number of states, while DeepDFA predicts DFAs with state counts comparable to the ground truth size. Moreover, our method proves to be the fastest among the four, except in a single case. We repeat the experiments on a corrupted version of the training datasets, where 1% of labels are erroneous. The results are presented in Table 4, showing the average performance over 10 trials. In this scenario, DFA-inductor is unable to provide a solution even for the smallest size DFAs. DeepDFA, L* extraction and DFA-generator (when it can converge) display only minor drops in performance, demonstrating resilience to noise. DeepDFA outperforms both the other stochastic methods in terms of test accuracy and state count prediction, and almost always employs the fewest number of parameters. **In summary**, the main issues encountered in competitor methods are: (i) scalability problems with DFA-inductor and DFA-generator, often resulting in out-of-memory or out-of-time errors when handling large DFAs. (ii) very low resilience to errors exhibited by DFA-inductor. (iii) tendency to significantly overestimate the automaton state space, possibly due to overfitting, observed in L* extraction. In contrast, our method consistently delivers the most precise results in terms of test accuracy and number of states. It achieves this without encountering any out-of-time or out-of-memory issues, while also demonstrating a competitive number of parameters and execution times across all experiments.

(a) Tomita1   (b) Tomita2   (c) Tomita3   (d) Tomita4

(e) Tomita5   (f) Tomita6   (g) Tomita7
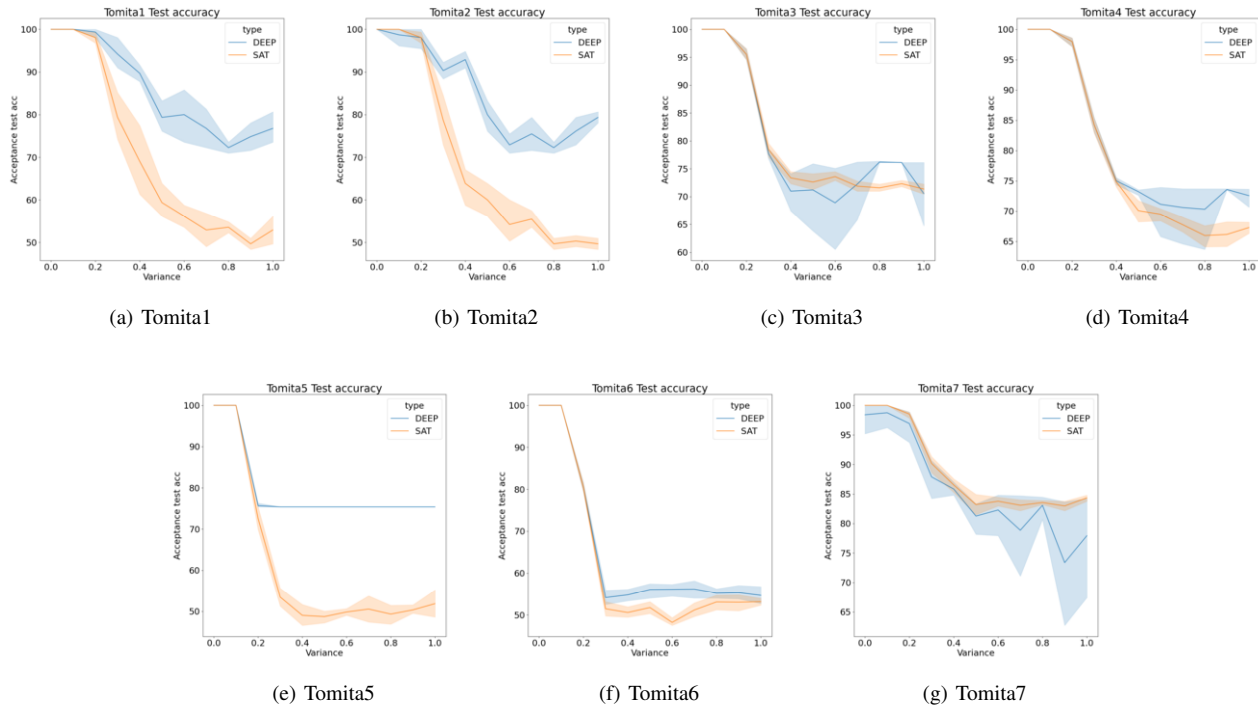
Figure 3: Learning DFA from traces composed of imperfectly grounded symbols: test accuracy on the 7 Tomita languages. In blue the results obtained with the extension of DeepDFA to probabilistic symbols, in orange results of DFA-inductor.

Learning DFAs from traces of imperfectly grounded symbols Automata learning typically takes a set of symbolic traces as input, which are sequences of propositional symbols grounded in a perfectly binary manner. However, in many domains, such for example Reinforcement Learning [23, 30], the observable sequences come with uncertainty in their grounding. Specifically, at each step of the trace, we have a set of probabilistic beliefs over the symbol set instead of a binary interpretation. To the best of our knowledge, this is the first approach to automata learning that can accommodate such 'probabilistic grounding' of symbols in the traces. To evaluate the capability of our framework in learning DFA specifications from sequences of imperfectly grounded symbols, we utilized a modified version of the dataset for Tomita Languages, where the one-hot representation of input symbols in the training traces is corrupted with Gaussian noise having a mean of zero and variable variance. We compared the extension of DeepDFA to handle probabilistic symbols (Equation 6) with DFA-inductor. Notably, boolean logic induction methods, such as DFA-inductor, cannot handle probabilistic truth values for the input symbols. Therefore, we discretized the corrupted inputs to the nearest one-hot vector before passing them to the method. Figure 3 shows that DeepDFA exhibits a greater resilience to noise in the input symbols than its competitor and its performance decrease smother as we increase the noise intensity.

Ablation study: the effect of changing the state space size RNNs have many hyperparameters and design choices that can affect performance: the model type, the number of layers, the number of features per layer, and more. By contrast, our recurrent neural model is a simplified structure with only one hyperparameter: the hidden state size. In this section, we discuss this hyperparameter choice. Figure 2(c-d) shows the effect on the test accuracy and the predicted number of states of tuning this hyperparameter. We conducted experiments using a random DFA with a size of 20 and an alphabet size of 3. The hidden state size was varied within the range of 10 to 120. Results

show that our model demands a hidden state size surpassing the actual number of states. The best performances for the random DFA of size 20, are achieved for $\hat{Q}_{max} \geq 70$—exceeding its actual state count by 3.5 times. Remarkably, the model maintains its robust performance even when the state size is markedly overestimated. Even at the largest state size, test accuracy remains commendably high, and the number of states inferred remains proximate to the actual count. This suggests that the model evades overfitting; despite being dimensioned to represent a substantially greater number of states, it leverages only a subset of them.

6 Conclusions and Future Work

In conclusion, we propose DeepDFA: a hybrid between a DFA and a recurrent neural network, which can be trained from samples with backpropagation as usual deep learning models, but that is completely interpretable, as a DFA, after training. Our approach takes the best from the two worlds: grammar induction on one side and recurrent neural networks on the other side. It uses fewer weights and only requires setting one hyperparameter compared to recurrent neural nets. At the same time, it can tolerate errors in the training labels and noise in the input symbols, and can be applied to large target DFAs, differently from exact methods. DeepDFA can find applications in various scenarios. In this paper, we present the framework in a general context, without specifying any particular application. However, we are keen to apply it in the area of non-Markovian Deep Reinforcement Learning [14, 26, 23]. Furthermore, we are developing a multi-layered version of DeepDFA, having the potential to expand its capabilities to even more intricate regular languages without compromising the explainability, which we leave for future research.

Acknowledgements

References

[1] S. Agostinelli, F. Chiariello, F. M. Maggi, A. Marrella, and F. Patrizi. Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114:102180, 2023.

[2] B. K. Aichernig, S. König, C. Mateis, A. Pferscher, D. Schmidt, and M. Tappler. Constrained training of recurrent neural networks for automata learning. In *Software Engineering and Formal Methods: 20th International Conference, SEFM 2022, Berlin, Germany, September 26–30, 2022, Proceedings*, page 155–172, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-17107-9.

[3] J.-C. Amengual, A. Sanchis, E. Vidal, and J.-M. Benedi. Language simplification through error-correcting and grammatical inference techniques. *Machine Learning*, 44(1/2):143–159, 2001.

[4] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. ISSN 0890-5401.

[5] T. R. Besold, A. d'Avila Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. In P. Hitzler and M. K. Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 1–51. IOS Press, 2021.

[6] S. Bhattamishra, K. Ahuja, and N. Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, Nov. 2020. Association for Computational Linguistics.

[7] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of finite-state machines for behavior control. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[8] M. Bugalho and A. L. Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recogn.*, 38(9): 1457–1467, sep 2005. ISSN 0031-3203.

[9] M. Collery, P. Bonnard, F. Fages, and R. Kusters. Neural-based classification rule learning for sequential data. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023. URL https://openreview.net/pdf?id=7tJyBmu9iCj.

[10] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 854–860. AAAI Press, 2013. ISBN 9781577356332.

[11] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005. ISSN 0031-3203. Grammatical Inference.

[12] W. De Mulder, S. Bethard, and M.-F. Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015. ISSN 0885-2308.

[13] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[14] M. Gaon and R. Brafman. Reinforcement learning with non-markovian rewards. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3980–3987, Apr. 2020. doi: 10.1609/aaai.v34i04.5814. URL https://ojs.aaai.org/index.php/AAAI/article/view/5814.

[15] A. d. Garcez and L. C. Lamb. Neurosymbolic ai: The 3rd wave, 2020. URL https://arxiv.org/abs/2012.05876.

[16] P. Grachev, I. S. Lobanov, I. Smetannikov, and A. Filchenkov. Neural network for synthesizing deterministic finite automata. *Procedia Computer Science*, 119:73–82, 2017.

[17] M. J. H. Heule and S. Verwer. Exact dfa identification using sat solvers. In *Proceedings of the 10th International Colloquium Conference on Grammatical Inference: Theoretical Results and Applications*, ICGI'10, page 66–79, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3642154875.

[18] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[19] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton, 1971.

[20] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[21] R. Kusters, Y. Kim, M. Collery, C. de Sainte Marie, and S. Gupta. Differentiable rule induction with learned relational features. In A. d'Avila Garcez and E. Jiménez-Ruiz, editors, *Proceedings of the 16th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 2nd International Joint Conference on Learning & Reasoning (IJCLR 2022), Cumberland Lodge, Windsor Great Park, UK, September 28-30, 2022*, volume 3212 of *CEUR Workshop Proceedings*, pages 30–44. CEUR-WS.org, 2022.

[22] K. J. Lang, B. A. Pearlmutter, and R. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI 1998: Grammatical Inference*, number 1433 in Lecture Notes in Computer Science book series (LNCS), pages 1–12. Springer, 1998. Cite as: Lang K.J., Pearlmutter B.A., Price R.A. (1998) Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar V., Slutzki G. (eds) Grammatical Inference. ICGI 1998. Lecture Notes in Computer Science, vol 1433. Springer, Berlin, Heidelberg.

[23] A. C. Li, Z. Chen, P. Vaezipoor, T. Q. Klassen, R. T. Icarte, and S. A. McIlraith. Noisy symbolic abstractions for deep RL: A case study with reward machines. *CoRR*, abs/2211.10902, 2022. doi: 10.48550/arXiv.2211.10902. URL https://doi.org/10.48550/arXiv.2211.10902.

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

[25] I. Salvador and J. Benedí. RNA modeling by combining stochastic context-free grammars and n-gram models. *Int. J. Pattern Recognit. Artif. Intell.*, 16(3):309–315, 2002.

[26] T. Shahar and R. I. Brafman. Reinforcement learning in rdps by combining deep RL with automata learning. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, pages 2097–2104, 2023. doi: 10.3233/FAIA230504. URL https://doi.org/10.3233/FAIA230504.

[27] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 440–449, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X.

[28] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108, Ann Arbor, Michigan, 1982.

[29] E. Umili and R. Capobianco. Deepdfa: Automata learning through neural probabilistic relaxations, 2024. URL https://arxiv.org/abs/2408.08622.

[30] E. Umili, F. Argenziano, A. Barbin, and R. Capobianco. Visual reward machines. In *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning, La Certosa di Pontignano, Siena, Italy, July 3-5, 2023*, pages 255–267, 2023.

[31] H. Walke, D. Ritter, C. Trimbach, and M. Littman. Learning finite linear temporal logic specifications with a specialized neural operator, 2021. URL https://arxiv.org/abs/2111.04147.

[32] Q. Wang, K. Zhang, A. Ororbia, X. Xing, X. Liu, and C. Giles. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Computation*, 30(9):2568–2591, Sept. 2018. ISSN 0899-7667. doi: 10.1162/neco_a_01111. Publisher Copyright: © 2018 Massachusetts Institute of Technology.

[33] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5247–5256. PMLR, 10–15 Jul 2018.

[34] I. Zakirzyanov, A. A. Shalyto, and V. I. Ulyantsev. Finding all minimum-size dfa consistent with given examples: Sat-based approach. In *SEFM Workshops*, 2017.

[35] I. Zakirzyanov, A. Morgado, A. Ignatiev, V. I. Ulyantsev, and J. Marques-Silva. Efficient symmetry breaking for sat-based minimum dfa inference. In *LATA*, 2019.