Machine Learning
https://doi.org/10.1007/s10994-021-05948-1

Distillation of weighted automata from recurrent neural networks using a spectral approach

Rémi Eyraud[1] · **Stéphane Ayache**[2]

Received: 18 September 2019 / Revised: 1 September 2020 / Accepted: 11 January 2021

Abstract
This paper is an attempt to bridge the gap between deep learning and grammatical inference. Indeed, it provides an algorithm to extract a (stochastic) formal language from any recurrent neural network trained for language modelling. In detail, the algorithm uses the already trained network as an oracle—and thus does not require the access to the inner representation of the black-box—and applies a spectral approach to infer a weighted automaton. As weighted automata compute linear functions, they are computationally more efficient than neural networks and thus the nature of the approach is the one of knowledge distillation. We detail experiments on 62 data sets (both synthetic and from real-world applications) that allow an in-depth study of the abilities of the proposed algorithm. The results show the WA we extract are good approximations of the RNN, validating the approach. Moreover, we show how the process provides interesting insights toward the behavior of RNN learned on data, enlarging the scope of this work to the one of explainability of deep learning models.

Keywords Weighted automata · Recurrent neural network · Spectral extraction · Grammatical inference · Explainability

1 Introduction

In his founding paper, Kleene (1956) studies the expressive power of the ancestors of our current neural networks, namely the nerve nets of McCulloch and Pitts (1943). To do so, he introduces two computational models that are now at the core of theoretical computer science: rational expressions and finite state automata. However, despite of this common

Editors: Olgierd Unold, François Coste, Colin de la Higuera.

✉ Rémi Eyraud
remi.eyraud@univ-st-etienne.fr

Stéphane Ayache
stephane.ayache@lis-lab.fr

[1] Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 Saint-Etienne, France

[2] Aix Marseille Univ, Universite de Toulon, CNRS, LIS, Marseille, France

Published online: 19 April 2021

origin, the fields of theoretical computer science and of deep learning look nowadays almost completely disjoint: the type of mathematics used, the pursued goals, and the followed research paths are rarely the same.

The context of the work presented in this article lies exactly within the frontier of these two fields and is thus an attempt to bridge this gap: it proposes a new way to link finite state machines and deep neural networks for sequential data.

When this kind of data is under consideration, deep learning approaches often rely on the use of Recurrent Neural Network (RNN). Indeed, these models are known to obtain state of the art learning results in various practical tasks, including speech processing (Graves et al. 2013), automatic natural language translation (Cho et al. 2014), or caption generation (Xu et al. 2015).

While a wide number of different RNN types exists (Lipton 2015; Salehinejad et al. 2018), they are all relying on the same mechanism: elements of the sequence are taken care of one at a time; for each element an output is computed; this output depends on the current element and of the part of the sequence seen so far. The memory mechanism used to remember the previously seen part is often based on the values outputted by the recurrent layer(s) on the previous element: this is usually a vector in a latent space that can be understood as the state of the network while dealing with the current element. The current output is thus computed using the current input and the value of the state-vector, making RNN *de facto* state machines. Their state space is infinite as it is a subset of $\mathbb{R}^n$, with n being an integer fixed by the architecture of the network (it corresponds usually to the size of the recurrent layer).

Despite their impressive learning ability in practice, RNN suffer from two main drawbacks. The first one is usually called the black-box problem: the process allowing a RNN to take a decision or to deal with a given task is not understandable by human beings. The computational use of a large amount of parameters and the combination of non-linear functions forbid the interpretation of its behavior. In other words, though the learning phase is likely to provide a trustworthy model, it brings no human knowledge about the problem and its resolution. Worst, in case of failure, no feedback can be inferred to provide ways to tackle the issue. One has thus to accept to blindly follow an opaque model...

One usual path followed to tackle this issue is called *local interpretability* (Guidotti et al. 2018). In this line of work, one aims at explaining how a neural network takes a decision on a particular datum. This usually relies on finding clues linking the decision to a particular part of the input (Ribeiro et al. 2016). For RNN, this could take for instance the form of highlighting the importance of each elements of the sequence based on its importance to the model's prediction (Li et al. 2016; Wallace et al. 2018).

But this approach is limited and it does not tackle the second drawback of RNN, that is, their computational cost. If it is nowadays admitted that a learning procedure could take hours or even days on state of the art computer grids, the use of the learned model should be computationally efficient. However, RNN may not have this property: their number of parameters grows quickly with their size and they are computing a composition of non-linear functions using these parameters. A consequence is for instance the wide use of cloud computing for cell phone applications requiring RNN (Chung et al. 2017), which at the same time forbids users with no or poor data connection to use them and is an ecological non-sense, in contradiction with green computing goals (Harmon and Auseklis 2009).

A way to overcome these two drawbacks, that are actually common to most deep artificial neural networks, is to achieve a *(knowledge) distillation* (Hinton et al. 2015; Furlanello et al. 2018), a process closely related to variational approximation (Metropolis and Ulam 1949). In this approach, a (deep) neural network is first learned, while in a second step a simpler

and less computationally costly model is extracted from the network. A well-known example of such procedure is the distillation of decision trees from usual feed-forward neural networks (Hayashi and Nakai 1990; Craven and Shavlik 1995; Schmitz et al. 1999; Frosst and Hinton 2017).

Hinton et al. (2015) have introduced the term distillation as the task of providing computationally efficient models by transferring knowledge from a complex architecture. This line of works usually focus on a teacher-student paradigm where a large "teacher" network guides the learning of a smaller "student" model. The knowledge transfer is typically achieved using a loss that aims at aligning outputs of the student with the ones of the teacher.

The models obtained by distillation usually enjoy one or both of two interesting features: they provide an easier *global interpretation* of the decision process while requiring less computational power. This constitutes the main motivation for the present article that aims at distilling finite state machines, namely Weighted Automata (WA), from already trained RNN. These models have been extensively studied (see Droste et al. (2009) for an overview), require only linear computations, and benefit from a graphical representation.

Our approach, whose preliminary details have been published in the proceedings of the International Conference on Grammatical Inference (Ayache et al. 2018), relies on an adaptation of the spectral learning algorithm for WA (Balle et al. 2014). While this approach requires a data set as input, our algorithm works from a trained RNN and uses it as an oracle, querying it well chosen sequences to get the value the network attributes to them. It then infers the automaton from a mathematical object called the Hankel matrix, storing the result of these queries.

The novelty of our distillation algorithm is mainly due to two factors: it does not require learning data and it does not need to access the inside of the RNN. This last point makes it adapted to work for any architecture, any type of cells, and even for black-boxes that are not RNN, providing that they are computing a function on sequential data.

This article firstly includes a survey on methods to distill finite state machines from RNN (Sect. 2). After carefully introducing notations and defining main notions in Sect. 3, we detail our approach in Sect. 4. The experimental framework is given in Sect. 5 and the results of the experiments in Sect. 6. Section 7 concludes this article.

2 Scientific context

In this section, we detail the main works that aim to bridge the gap between RNN and the models used in classical computer science, those of the formal language theory.

Though more details are given in Sect. 3, we need first to recall the formalization of the general behaviour of RNN: at time-step t, that is, on the tth element on the input sequence, the output of the recurrent layer(s) is given by:

$$h_t = \mathcal{R}(x_t, h_{t-1})$$

where x_t is the tth element of the sequence, and $\mathcal{R}$ the recurrent or update function. As already stated, the vector h_t is the value of the state of the RNN after seeing the tth element.

2.1 Relation of RNN classes to other computational models

When the expressive power of RNN is under consideration, one usually refers to the work of Siegelmann and Sontag (1992, 1994): it shows that Simple Recurrent Neural Networks

(SRNN) (Elman 1990) are Turing complete, that is, that any function computable using a Turing machine can be computed using a SRNN. These networks deserve their name: the update function for the hidden state correspond to applying an activation function f to the weighted sum of the previous state and the current input:

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

where W and U are weights matrices, and b is a scalar.

As impressive as this result might look, it is of little help in practice: the proof requires to have rational weights of infinite precision, an unbounded computation time, and relies on a mechanism allowing the network to "think" before computing an output. This is far from corresponding to the networks used nowadays. In addition, not all recurrent neural networks can be learned using a gradient descent algorithm, the approach currently used for this task. However, few papers have tackled the more practical question of the characterization of the classes of learnable finite precision and time neural nets.

One exception is the work of Rabusseau et al. (2019) where the authors show an equivalence between Weighted Automata (WA) (detailed in Sect. 3.3) and linear second order RNN. These models are defined as follow:

$$h_t = f(T \cdot x_t \cdot h_{t-1})$$

where T is a tensor of order 3.

The obtained result also asserts that the number of states in the minimal WA computing a given function is exactly the number of neurons in the hidden layer of the RNN for the same function, adding a structural equivalence to the expressivity one. In addition, the paper shows how a learning algorithm for WA can be adapted to this type of RNN, keeping its proven learning capacity.

Another recent exception is the work of Weiss et al. (2018b) that compares different recurrent network architectures. It empirically shows that finite precision SRNN and Gated Recurrent Units (GRU) (Cho et al. 2014) cannot compute non-regular recognizers, while Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) can be trained to recognize some context-free and even some context-sensitive dependencies. The paper also proposes a connection between LSTM and a simplified version of counter automata (Hopcroft and Ullman 1979).

Following this work, Merrill (2019) focuses on the kinds of computation that real-time, bounded-precision neural networks can perform by relating them to different types of automata. In this theoretical work, different types of RNN (and CNN) are studied in a framework the author called *asymptotic convergence*: their expressiveness is considered at pointwise convergence (Rudin 1976). In this context, the space state of RNN tends to be discrete and the following results can be proven: LSTM behave similarly to counter machines; the class of SRNN-acceptable and GRU-acceptable languages is exactly the regular languages. Empirical studies partially confirm these results, though some experiments show a clear limit to the asymptotic framework by contradicting some of the obtained theoretical results. A recent direct continuation of this work (Merrill et al. 2020) renames the framework into *saturated RNN* study, adds to state expressiveness the notion of language expressiveness,[1] and propose a hierarchy based on this elements covering a large amount of RNN classes.

[1] While state expressiveness focuses on the variability that can be obtained within the latent space of a type of RNN, language expressivity deals with the languages potentially recognizable from this vector when using different functions that map it to a Boolean.

Adopting a different research path, Avcu et al. (2017) investigate the learning ability of RNN on different sub-regular classes of languages: their experiments show that long term dependencies are not always learned by RNN, even by gated formalism like LSTM. Mahalunkar and Kelleher (2019) recently continue this line of work and confirm its conclusions on large benchmarks.

Other approaches have been proposed for this goal of studying RNN in relation to other computational models. For instance, it is worth to cite the work of Chen et al. (2017). Though it does not link RNN to another computational model *per se*, it aims at studying single-layer, ReLU-activation, rational-weight RNN using usual formal language approaches. It concludes that classical properties of computational models, such as consistency, equivalence, and minimization, are not decidable for this type of RNN, while these problems are known to be tractable for models like WA.

Finally, in a recent article, Marzouk and de la Higuera (2020) adopt also the classical approach of theoretical computer science to provide insights on the link between RNN and several classes of stochastic finite state machines. They show in particular that the equivalence problem of classic classes of finite state machines and weighted first-order RNN is not decidable in general. They also show how the problem becomes decidable when restricting the study to particular (and reasonable) RNN sub-classes, though its complexity remains intractable (NP-hard).

2.2 Distillation of binary classifiers on sequential data

First ideas for knowledge distillation from RNN concentrate on binary classifiers, sometime called RNN-acceptors. Indeed, early works train a RNN on data coming from a Deterministic Finite State Automaton (DFA) and then propose an algorithm to extract a DFA from the RNN. As DFA are language models, the output of the learned RNN consist of 2 possible values: $+1$ if the complete input sequence is in the language, -1 (or 0) otherwise.

Though first attempts consist in feeding a RNN with training data coming from a DFA and then using a test set to determine if it learned correctly [see for instance the work of Cleeremans et al. (1989) and Watrous and Kuhn (1992)], some of these works already focus on distilling a DFA from RNN.

To our knowledge, the first attempt at this task is the work of Giles et al. (1992) where the authors use a quantization algorithm on the state space of second order RNN. This approach tries to partition the RNN state space by dividing each dimension into equal intervals. If this was tractable with the RNN used in the 90's, it clearly is not with today networks, given the explosion of the dimension of the internal state.

Another idea is to use a clustering algorithm on the state space of RNN. Though Cleeremans et al. (1989) are already using hierarchical clustering to verify that the structure of the RNN state space corresponds to the one of the target DFA, distillation of DFA using that approach is first introduced by Giles et al. (1992) and enjoys many refinements (see for instance the work of Manolios and Fanelli (1994), Omlin and Giles (1996), or Cechin et al. (2003)). As in the previous approach, data is fed to a RNN already trained on data coming from a DFA and the vector values obtained all along the different input sequences is stored. Then a clustering algorithm, like the k-means one (Steinhaus 1957), is used on these vectors to get k clusters, each one corresponding to a state. The transitions between states are then obtained by parsing selected sequences into the RNN to get the cluster the resulting vector belongs to.

These works are the starting point of numerous refinements, most of which are detailed in the survey of Jacobsson (2005) and empirically compared on different RNN architectures in a recent paper by Wang et al. (2017).

Weiss et al. (2018a) recently open a new approach for the distillation of DFA from RNN: they adapt the L* algorithm from Angluin (1987) so that the RNN plays the role of the oracle. The algorithm first completes a table whose rows correspond to prefixes and columns to suffixes by querying the RNN to know which sequence made of one given prefix and one given suffix is recognized by the RNN. Then a DFA is inferred from the table and the algorithm has to decide whether this DFA is equivalent to the RNN, a problem recently shown to be undecidable (Marzouk and de la Higuera 2020). While in the initial theoretical framework this step is straightforward, it requires carefully defined heuristics when distilling from a RNN: the authors propose an iterative process that parse at the same time the distilled DFA and another one obtained by partitioning the RNN state space using SVM. The latter is refined recursively until one can conclude the RNN and the DFA are equivalent, or a sequence on which the distilled DFA and the RNN disagree is obtained. If this last event occurs, the process continues since the obtained counter example allows modifications of the table and thus the distillation of a better DFA. Their experiments show that their algorithm distills in an efficient way smaller and more accurate DFA than previous methods, allowing a better understanding of the behaviour of the RNN.

To be complete on the subject, one should cite the work of Wang et al. (2019a) where the authors extract DFA from different types of RNN in order to find adversarial examples. Their experiments show that such examples can be exhibited for almost all architectures, the type of examples challenging the robustness depending on the type of RNN considered.

2.3 Distillation of estimators

While of great interest, distillation of DFA from neural networks suffers from one important drawback: it does not correspond to the practical use of RNN. Indeed, these models are rarely trained for binary classification. While other practical frameworks exist (see for instance the work of Cho et al. (2014)), RNN are mainly trained for *language modeling*, a setting in which the models are trained to guess the probability of each potential symbol to be the next in the sequence (a particular symbol often marks the end of sequence). After each new input symbol x_t, these Language Modelling RNN (LM-RNN) thus output a vector of real values, each value corresponding to the estimated probability of a given symbol to be the following one in the sequence.

As far as we know, first attempts at such a distillation process take place in the context of speech recognition, where having computationally efficient model is critical. For instance, the work of Deoras et al. (2011) proposes to sample a LM-RNN and then infers a n-gram from the sampled sequences. Experiments show better results than just inferring n-gram from training sample of usual NLP benchmarks.

A second approach, still in the context of early phases of speech recognition, is the work of Lecorvé and Motlícek (2012). Their algorithm aims at distilling Weighted Automata from LM-RNN: they use the k-means algorithm to cluster vectors that consist of the concatenation of vectors from the state space of the network and the last symbol seen. Their approach refines iteratively this discretisation in a greedy manner and massively uses domain specific heuristics to avoid the explosion of the size of the distilled WA and to allow back-tracking. Experiments show globally better results than the previously cited approach using n-gram.

In a recent article, Okudono et al. (2019) extended the work on DFA of Weiss et al. (2018a) to WA, focusing on RNN assigning a single real value to sequences. Instead of having a binary table, they store the weights of the sequence made of the concatenation of the prefix defining the row and the suffix of the corresponding column. To complete the table, they replace the idea of distinct row values by the notion of co-linearity. Then, using Balle and Mohri (2015) algorithm, they distill a WA and, finally, need to check its (approximate) equivalence with the RNN. To achieve this last goal, they open the black-box and try to iteratively learn a regression function from the observed states of the RNN to WA configurations, that is, the real value vectors one can obtain along the parsing of a sequence into a WA (see Sect. 3.3 for details). If they exhibit such function, then the WA and RNN are considered equivalent, up to a predefined threshold. If not, the process gives them a counter example, and the algorithm continues in the same way than in the DFA distillation case.

Finally, Weiss et al. (2019) have also recently extended their approach to distill deterministic weighted automata from LM-RNN. Their key ideas are to use conditional probabilities to fill the table and a tolerance hyper-parameter to distinguished the states from the table, as co-linearity is a too strong requirement, especially in the presence of noise. Reported empirical results are promising though the method requires some heuristics to be run in reasonable time and the obtained automata can consist of thousands of states.

3 Preliminaries

3.1 Elements of language theory

In theoretical computer science, a finite set of symbols is called an *alphabet*, usually denoted by the Greek letter Σ. Language theory mainly deals with finite sequences on an alphabet that are called *strings* and are usually denoted by the letters w, v, or u. We denote the set of all possible strings over Σ by Σ^*. For instance, the alphabet can be the ASCII characters, the 4 nucleobases of DNA, Part-of-Speech tags or lemmas from Natural Language Processing, or even a set of symbols obtained by the discretization of a time series (Dimitrova et al. 2010).

Throughout the paper, we will use other notions from language theory: the *length* of a string w is the number of symbols of the sequence (denoted $|w|$); the string of length zero is denoted λ; given 2 strings u and v we note uv their concatenation; if a string w is the concatenation of strings u and v, $w = uv$, we say that u is a *prefix* of w and that v is a *suffix* of w. Given a set of strings S, we denote pref(S) the set of all the prefixes of the elements of S.

3.2 Functions on sequences

In this article we consider functions that assign real values to strings: $f : \Sigma^* \to \mathbb{R}$. These functions are known under the name of *series* (Sakarovitch 2009). In particular, probability distributions over strings are such functions. Each of these functions is associated with a specific object that has been proven to be extremely useful:

Definition 1 (Hankel Matrix Berstel and Reutenauer 1988) Let f be a series over Σ^*. The Hankel matrix of f is a bi-infinite matrix $\mathcal{H} \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ whose entries are defined as $\mathcal{H}(u, v) = f(uv)$, $\forall u, v \in \Sigma^*$. Rows are thus indexed by prefixes and columns by suffixes.

For obvious reasons, only finite sub-blocks of Hankel matrices are of interest. An easy way to define such sub-blocks is by using a *basis* $\mathcal{B} = (\mathcal{P}, \mathcal{S})$, where $\mathcal{P}, \mathcal{S} \subseteq \Sigma^*$. If we note $p = |\mathcal{P}|$ and $s = |\mathcal{S}|$, the sub-block of $\mathcal{H}$ defined by $\mathcal{B}$ is the matrix $H_{\mathcal{B}} \in \mathbb{R}^{p \times s}$ with $H_{\mathcal{B}}(u, v) = \mathcal{H}(u, v)$ for any $u \in \mathcal{P}$ and $v \in \mathcal{S}$. We may write H if the basis $\mathcal{B}$ is arbitrary or obvious from the context.

3.3 Weighted automata

The following definitions are adapted from Mohri (2009) and Balle et al. (2014):

Definition 2 (Weighted automaton) A weighted automaton (WA) is a tuple $\langle \Sigma, Q, \mathcal{T}, \gamma, \rho \rangle$ such that:

- Σ is a finite alphabet;
- Q is a finite set of states;
- $I \subseteq Q$ is the set of initial states;
- $\mathcal{T} : Q \times \Sigma \times Q \to \mathbb{R}$ is the transition function;
- $\gamma : Q \to \mathbb{R}$ is an initial weight function;
- $\rho : Q \to \mathbb{R}$ is a final weight function.

A transition is usually denoted (q_1, σ, p, q_2) instead of $\mathcal{T}(q_1, \sigma, q_2) = p$. We say that two transitions $t_1 = (q_1, \sigma_1, p_1, q_2)$ and $t_2 = (q_3, \sigma_2, p_2, q_4)$ are consecutive if $q_2 = q_3$. A path π is an element of $\mathcal{T}^*$ made of consecutive transitions. We denote by $o[\pi]$ its origin and by $d[\pi]$ its destination. The weight of a path is defined by $\mu(\pi) = \gamma(o[\pi]) \times \omega \times \rho(d[\pi])$ where ω is the multiplication of the weights of the constitutive transitions of π. We say that a path $(q_0, \sigma_1, p_1, q_1) \dots (q_{n-1}, \sigma_n, p_n, q_n)$ reads a string w if $w = \sigma_1 \dots \sigma_n$.

The weight of a string w, that is, the real value assigned by the WA to w, is the sum of the weights of the paths that read w. A weighted automaton assigns weights to strings, that is, it maps each string of Σ^* to a real value. The set of functions computable by a WA is called the *rational series*.

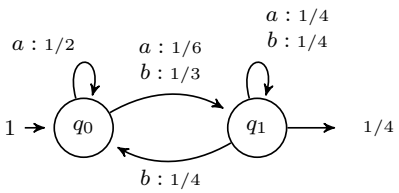
WA admit an equivalent representation using linear algebra:

Definition 3 (Linear representation Denis and Esposito 2008; Balle et al. 2014) A *linear representation* of a WA A is a triplet $\langle \alpha_0, (M_\sigma)_{\sigma \in \Sigma}, \alpha_\infty \rangle$ where

- the vector α_0 provides the initial weights (*i.e.* the values of the function γ for each state),
- the vector α_∞ is the terminal weights (*i.e.* the values of function ρ for each state),
- each matrix M_σ corresponds to the σ-labeled transition weights ($M_\sigma(q_1, q_2) = p \iff \mathcal{T}(q_1, \sigma, q_2) = p$).

The dimension of each element is the number of states of the automaton.

Figure 1 shows the same WA using the two representations. In what follows, we will consider that WA are defined in terms of linear representations, unless specified otherwise.



$$\alpha_0 = \begin{bmatrix}1\\0\end{bmatrix} \quad \alpha_\infty = \begin{bmatrix}0\\1/4\end{bmatrix}$$

$$M_a = \begin{bmatrix}1/2 & 1/6\\0 & 1/4\end{bmatrix} \quad M_b = \begin{bmatrix}0 & 1/3\\1/4 & 1/4\end{bmatrix}$$

Fig. 1 A WA in its graphical form and its equivalent linear representation

To compute the weight that a WA A assigns to a string $w = \sigma_1\sigma_2 \dots \sigma_n$ using a linear representation, it suffices to compute the product $A(w) = \alpha_0^\top M_w \alpha_\infty = \alpha_0^\top M_{\sigma_1} M_{\sigma_2} \dots M_{\sigma_n} \alpha_\infty$.

This can be interpreted as a projection into $\mathbb{R}^r$, where r is the number of states of A, followed by an inner product (Rabusseau et al. 2019). Indeed, $\alpha^0 = \alpha_0^\top$ is a vector that corresponds to the initial projection; each of the following steps computes a new vector α^i in the same space, moving from one vector to the next one by computing the product with the corresponding symbol matrix ($\alpha^i = \alpha^{i-1} M_{\sigma_i}$); the final projection is $\alpha^n = \alpha_0^\top M_w$; the output of the WA on w is given by the inner product $\langle \alpha^n, \alpha_\infty \rangle$.

α^i is sometime referred to the WA configuration after seeing the prefix $\sigma_1 \dots \sigma_i$. It is a vector of the latent state space of WA since the output of the models depends of this vector and the following symbols, in a standpoint analogue to the one of RNN. This is of crucial importance to understand WA: their state space is potentially infinite as it is a part of $\mathbb{R}^r$. It also distinguishes these models from classically used others, like DFA—whose state space is discrete finite—and in some sense Hidden Markov Models (HMM)—which model distributions over a discrete finite set of states.

If a WA computes a probability distribution over Σ^*, it is called a stochastic WA. In this case, it can easily be used to compute the probability of each symbol to be the next one of a given prefix (Balle et al. 2014): the probability of σ being the next symbol of the prefix w is given by

$$\alpha_0^\top M_w M_\sigma \tilde{\alpha}_\infty = \alpha^{|w|} M_\sigma \tilde{\alpha}_\infty$$

where $\tilde{\alpha}_\infty = (Id - (\sum_{\sigma \in \Sigma} M_\sigma))^{-1} \alpha_\infty$, with Id the identity matrix.

The following theorem is at the core of the spectral learning of WA (introduced independently by Hsu et al. (2009) and Bailly et al. (2009)) and of our approach (described in Sect. 4):

Theorem 1 (Carlyle and Paz 1971; Flies 1974) *A function $f : \Sigma^* \to \mathbb{R}$ can be defined by a WA if and only if the rank of its Hankel matrix is finite. In that case this rank is equal to the minimal number of states of any WA computing f.*

3.4 Recurrent neural networks

Recurrent Neural Networks (RNN) are artificial neural networks designed to handle sequential data. To do so, the RNN we consider incorporates an internal state that is used as memory to take into account the influence of previous elements of the sequence when computing the output for the current one.

Two type of architecture units are mainly used: the widely studied Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and the recent Gated Recurrent Unit (GRU) (Cho et al. 2014). In both cases, these models realize a non-linear projection of the seen prefix into $\mathbb{R}^d$, where d is the number of neurons on the penultimate layer: this vector is usually called latent representation of the part of the sequence seen so far. The last, usually dense, layer — several layers can potentially be used — specializes the RNN to its targeted task from this final latent layer.

A RNN is often trained to perform the next symbol prediction task[2]: given a prefix of a sequence, it outputs the probabilities for each symbol to be the next symbol of the sequence (a special symbol denoted $\rtimes$ [resp. $\ltimes$] is added to mark the start [resp. the end] of a sequence). Outputting the final symbol means that the input sequence is not a prefix of a longer sequence but instead a complete sequence. Notice that it is easy to use such RNN to compute the probability given to a string $w = \sigma_1\sigma_2 \dots \sigma_n$: one only needs to compute

$$P(w) = P(\rtimes)P(\sigma_1|\rtimes)P(\sigma_2|\rtimes\sigma_1)\dots P(\ltimes|\rtimes\sigma_1\sigma_2\dots\sigma_n)$$

4 Algorithm

In this section, we describe our approach: from an already trained LM-RNN, we fill a finite part of the Hankel matrix, then use a spectral approach to obtain a WA from the matrix. The following subsections detail the different steps of our proposition

4.1 From Hankel to WA

The Proof of Theorem 1 is constructive: it provides a way to generate a WA from its Hankel matrix $\mathcal{H}$. Moreover, the construction can be used on particular finite sub-blocks of this matrix: the ones defined by a complete and prefix-closed basis. Formally, a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is *prefix-closed* iff for all $w \in \mathcal{P}$, all prefixes of w are also elements of $\mathcal{P}$; $\mathcal{B}$ is *complete* if the rank of the sub-block $H_\mathcal{B}$ is equal to the rank of $\mathcal{H}$.

Explicitly, from such sub-block $H_\mathcal{B}$ of $\mathcal{H}$ of rank r, one can compute a minimal WA using a rank factorization $PS = H_\mathcal{B}$, with $P \in \mathbb{R}^{p\times r}$, $S \in \mathbb{R}^{r\times s}$. Let us denote:

- H_σ the sub-block defined over $\mathcal{B}$ by $H_\sigma(u, v) = \mathcal{H}(u\sigma, v)$,
- $h_{\mathcal{P},\lambda}$ the p-dimensional vector with coordinates $h_{\mathcal{P},\lambda}(u) = \mathcal{H}(u, \lambda)$ ($h_{\mathcal{P},\lambda}$ is the first column of $H_\mathcal{B}$ if suffixes are given in lexicographic order),
- $h_{\lambda,\mathcal{S}}$ the s-dimensional vector with coordinates $h_{\lambda,\mathcal{S}}(v) = \mathcal{H}(\lambda, v)$ ($h_{\lambda,\mathcal{S}}$ is first line of $H_\mathcal{B}$ if prefixes are sorted in lexicographic order).

The WA $A = \langle \alpha_0, (M_\sigma)_{\sigma\in\Sigma}, \alpha_\infty \rangle$, with:

$$\alpha_0^\top = h_{\lambda,\mathcal{S}}^\top S^+, \alpha_\infty = P^+ h_{\mathcal{P},\lambda}, \text{ and, for all } \sigma \in \Sigma, M_\sigma = P^+ H_\sigma S^+,$$

[2] This task is often called *language modelling* and the corresponding networks are denoted LM-RNN.

is a minimal WA[3] whose Hankel matrix is exactly the initial one (Balle et al. 2014).

This procedure is the core of the theoretically founded spectral learning algorithm (Bailly et al. 2009; Hsu et al. 2009; Balle et al. 2014), where the content of the sub-blocks is estimated by counting the occurrences of strings in a learning sample. Contrary to that, the work presented here uses an already trained RNN to fill $H_{\mathcal{B}}$ and H_σ on a carefully selected basis $\mathcal{B}$.

4.2 Proposed algorithm

Our algorithm is presented in synthetic form in Algorithm 1 . It can be broken down into three steps: from a LM-RNN model trained on some data, denoted $\mathcal{M}$, we first build a basis $\mathcal{B}$; second, we fill the required sub-blocks $H_{\mathcal{B}}$ and $(H_\sigma)_{\sigma\in\Sigma}$ with the values computed by the initial RNN $\mathcal{M}$; and third, we extract a WA from the Hankel matrix sub-blocks by low-rank factorisation of $H_{\mathcal{B}}$.

Algorithm 1: Extraction of a WA from an already trained RNN

Input : RNN $\mathcal{M}$, p, s numbers of prefixes and suffixes, r rank approximation
Output: A, a Weighted Automaton
$(\mathcal{P},\mathcal{S}) \leftarrow Generate\_Basis(\mathcal{M},p,s)$;
$H_{\mathcal{B}},(H_\sigma)_{\sigma\in\Sigma} \leftarrow Fill\_Hankels(\mathcal{M},\mathcal{P},\mathcal{S})$;
$A \leftarrow Spectral\_Extraction(H_{\mathcal{B}},(H_\sigma)_{\sigma\in\Sigma},r)$; `// using equations from 4.1`
return A;

4.2.1 Generating basis

Choosing the right basis $\mathcal{B} = (\mathcal{P},\mathcal{S})$ to define the right sub-block is an important task and different possibilities have been studied in the context of spectral learning (Bailly 2011; Quattoni et al. 2017).

A first idea is to implement *Generate_Basis*() by sampling from the uniform distribution on symbols with a maximum length parameter. A second possibility, if the black box is a generative device like LM-RNN, is to use it to build a basis, for instance by recursively sampling a symbol from the next symbol distribution given by the network.

Though other ways of generating a basis can be designed, for instance by using training data if available, the experiments presented in this paper concentrate on these two approaches, in Sect. 6.2.

Once a string is obtained, we add all its prefixes to $\mathcal{P}$ (to be prefix-closed) and all its suffixes to $\mathcal{S}$. The process is reiterated until $|\mathcal{P}| \geq p$. If needed, the set of suffixes is then completed in the same way until $|\mathcal{S}| \geq s$.

4.2.2 Filling Hankel matrix

Once we have a basis $\mathcal{B}$, the procedure *Fill_Hankels*() uses the black box RNN to compute the content of the sub-blocks $H_{\mathcal{B}}$ and $(H_\sigma)_{\sigma\in\Sigma}$: it queries each string made of selected prefix and suffix to the black box and fill the corresponding cells in the sub-blocks of the Hankel matrix with its answer. This procedure is the core step for knowledge distillation

[3] As usual, N^+ denotes the Moore–Penrose pseudo-inverse (Moore 1920) of a matrix N.

in our algorithm, enabling the knowledge from RNN outputs to be directly transferred into the Hankel matrix from which we can extract WA by spectral extraction.

4.2.3 Spectral extraction

Finally, a rank factorization $H_{\mathcal{B}} = PS$ for a certain rank parameter r, has to be obtained: the function *Spectral_Extraction*() performs a Singular Value Decomposition on $H_{\mathcal{B}}$ and truncates the result to obtain the needed rank factorization (see (Balle et al. 2014) for details). It then generates a WA using the formulas described in Sect. 4.1. Section 6.5 elaborates on the top highest singular values and their relations to number of states of WA.

5 Experimental framework

5.1 Data

When one wants to systematically studies an algorithm working on sequences, there does not exist a wide number of benchmarks not specific to a particular field of application and thus that does not require large and impacting pre-treatment (for instance, NLP corpus require important and result influencing pre-processing steps, *e.g.*, POS-tagging, lemmatisation, etc.). As a consequence, researchers tend to compare their work using well-known problems from the early times of machine learning, mostly the Reber's grammar (Reber 1967) and the Tomita problems (Tomita 1982) (see for instance the work of Weiss et al. (2018a), Mozer et al. (2018), or Wang et al. (2019b) for recent examples of uses of these data).

Though of great interest, these two sets of problems are too specific and neither large nor covering enough to convincingly support claims about an algorithm only tested on them. Fortunately, 2 benchmarks have been recently made available that allow an as exhaustive as possible study of the type of algorithms we are interested in. Each of them were made freely available in the context of on-line challenges : the PAutomaC (Verwer et al. 2014) and the SPiCe (Balle et al. 2017) competitions.

5.1.1 PAutomaC data

The first sets of data we experiment on are the ones of the competition PAutomaC whose goal was to learn distributions from multi-sets of sequences of symbols of varying length. The competition provides 48 artificial problems generated by randomly generated finite state machines : Deterministic Probabilistic Finite Automata (DPFA), non-deterministic Probabilistic Finite Automata (PFA), and Hidden Markov's Models (HMM). Large range of machine sizes, alphabet sizes, and sparsity indicators values are covered by the different instances and the competition results proved it is covering a wide range of difficulty levels.

Though all of these models are strictly less expressive than stochastic WA (Denis and Esposito 2008), it is relevant to test our distillation procedure on this wide variety of problems. Moreover, having access to the true target model gives an interesting point of comparison with the extracted automaton.

For each of these 48 problems, we have access to a training set (20,000 or 100,000 sequences), that we only use to train the RNN.

5.1.2 SPiCe data

The SPiCe benchmark is made of 15 problems covering a large variety of contexts, from various natural languages processing data to software engineering data, including bioinformatics and well-chosen synthetic ones.

The aim of the competition was to learn a model from each of the available training samples and then to propose a ranking of the top 5 next potential symbols to given prefixes.

These test sets made of prefixes are not what we need for our task, since they do not provide complete information on whole sequences. We thus split randomly the available learning samples into a training and a test sample, keeping 20% for the test.

We also choose to not handle problem 11 of SPiCe, due to its large vocabulary (6722 symbols) that would require long computation times for each of the parameters we want to study.

5.2 RNN training

We base the architecture on the work of Shibata and Heinz (2017), who won the SPiCe competition, itself inspired by Sutskever et al. (2014). The architecture is quite simple: it is composed of an initial embedding layer (with $3 * |\Sigma|$ neurons), two GRU (or LSTM) layers, one or two dense layers, followed by a final dense layer with `softmax` activation composed of $|\Sigma| + 1$ neurons. The networks are trained using the back-propagation through time algorithm (Rumelhart et al. 1986; Werbos 1988) with the categorial cross-entropy loss function (Rubinstein and Kroese 2004) on the distribution over next symbols for all prefixes.

Given this framework, we considered several hyper-parameters to tune. First, the number of neurons in the recurrent layers and the following dense layer: we tested a number of neurons between 30 and 400, the first following dense layer uses half of it, the second dense layer is set as the size of the input embedding layer, or ignored. We also compared `tanh` and `RELu` activations. We trained our networks during 150 epochs. We do witness expected over-fitting before this limit: after a phase in which the loss on a validation set decreases, it starts to rise. This confirms that this number is adequate as maximum iterations.

We finally selected one single configuration for each problem that scores the best categorical cross-entropy on the validation set in average since we did not observe significant variations among tested configurations. The results detailed below are obtained from the following configuration: an initial embedding layer, then two GRU (or LSTM) layers build with 400 neurons and `tanh` activations; followed by one hidden dense layer with 200 neurons and `RELu` activations, before the final output layer. We trained models using Adam optimizer for 150 epochs with batch size of 512, a learning rate of 0.001, and defaults beta values.

The evaluation of this protocol shows good learning results but it is also clear that better results could be obtained tuning the RNN on the chosen data independently for each problem. We decide to not push to its limits this learning part since it is not central in this work. Moreover, having RNN with various learning qualities is interesting from the standpoint of the evaluation of the WA extraction: we expect the WA to be as good—or, thus, as bad—as the RNN. We report in appendix the validation loss for each RNN model.

5.3 Metrics

5.3.1 Evaluation samples

In order to not bias the evaluation by picking a particular type of data, we use two evaluation sets to compute the different metrics chosen to evaluate the quality of the WA extraction. S_{Test} is the test set from the competition: directly the one used for PAutomaC, and the one coming from a random partitioning of the learning sample for SPiCe. S_{RNN} consists in generated sequences sampled using the learned RNN. Both S_{Test} and S_{RNN} contain 1000 sequences.

This way we test the quality of our distillation process using the distribution of initial learning task and the one of the RNN we are targeting. Notice that S_{RNN} is more relevant for our task: it contains sequences enjoying high probability in the model we are distilling from. S_{Test} has this property only if the RNN achieves good learning results.

We evaluate the quality of the approximation obtained by the extraction using two metrics.

5.3.2 Normalized discounted cumulative gain

The second metrics is the Normalized Discounted Cumulative Gain (NDCG, used in SPiCe). This metric is given by: for a prefix w of a sequence in an evaluation set,

$$\text{NDCG}_n(w, \widehat{\sigma}_1, ..., \widehat{\sigma}_n) = \frac{\sum_{k=0}^{n} \frac{P_{\text{RNN}}(\widehat{\sigma}_k|w)}{log(k+1)}}{\sum_{k=0}^{n} \frac{P_{\text{RNN}}(\sigma_k|w)}{log(k+1)}}$$

where σ_k is the k$^{\text{th}}$ most likely next symbol of w following P_{RNN} and $\widehat{\sigma}_k$ is the k$^{\text{th}}$ most likely next symbol of w following P_{WA}.

The NDCG_n score of an extraction is the sum of NDCG_n on each prefix of a sequence in the evaluation set, normalized by the number of these prefixes in the test set. In our experiment, we focus on NDCG_5 scores: this value for n is the one used for the SPiCe competition and it is actually the largest computable on all datasets, since the smallest alphabet size is 4 (to which a symbol marking the end of a sequence is added).

5.3.3 Word error rate for distillation

Word Error Rate (WER) is a classical quality measurement in the field of Natural Language Processing. In its simplest and most used form, it is computed from a set of sequences by counting on all prefixes the number of times the most probable next symbol in the model is different to the actual next symbol in the sequence. This total number is then divided by the total number of prefixes in order to have a value between 0 and 1, the closer to 0 the better.

Given that we have access to two evaluation samples (see Sect. 5.3.1), a simple idea would be to compute the classical WER on these sets. However, in the context of distillation we can design a different metric, inspired by WER, that will better suit the evaluation of our goal.

The idea is to use the black-box as ground truth: instead of looking at the next symbol in the dataset, one can compare the most probable next symbol in the distilled model and the most probable one in the model from which the distillation occurred. We call this metric Word Error Rate for Distillation (WER-D). Formally:

$$\text{WER-D}_S(\text{RNN}, \text{WA}) = \frac{1}{|\text{pref}(S)|} \sum_{w \in \text{pref}(S)} \mathbb{1}_{\sigma_1(w) \neq \widehat{\sigma}_1(w)}$$

where S is an evaluation set and, for any prefix w, $\sigma_1(w)$ is the most probable next symbol in the RNN and $\widehat{\sigma}_1(w)$ the one in the WA.

WER-D is thus the frequency observed on a test set of the disagreement between the two models on the next most probable symbol. Though this metric has not been described in any article yet, as far as we know, it is the one used by Weiss et al. (2019), as their available code shows.

5.4 Experimental setting

We describe here the protocol we followed to validate our method for RNN distillation. We also detail the experiments we conducted and hyper-parameters we explored for WA extraction.

5.4.1 Global procedure

We first trained RNN with the different architecture parameters detailed in previous section on the 62 benchmark problems of the PAutomaC and SPiCe competitions. We then keep the best RNN in term of categorical cross-entropy for each problem and consider it as the black-box of the problem.

We then extract WA from this RNN following the algorithm described in Sect. 4. We test both already described approaches for basis generation (uniform distribution on symbols and RNN sampling) with a wide range of parameter values:

- the size of the basis varies from 100 prefixes and suffixes up to 3000,
- the rank for the Hankel matrix factorization is also taken between 1 and the full rank of Hankel matrix (up to 3000).

5.4.2 Hyper-parameters for extraction

We compare different hyper-parameter values for the extraction algorithm: size p and s of the basis is varying between (100, 100) and (3000, 3000). We tested different rank values from 1 to the number of prefixes. On Spice datasets, we considered all rank values below 15 and 25 others logarithmically picked in the other possible values. On PAutomaC datasets, we tested 20 ranks below 50 then 5 ranks logarithmically chosen until the number of prefixes. We also compare two possible strategies for sampling the basis to build the Hankel matrix, as discussed in Sect. 4.2.1.

Some values of the size of the basis exceed what is reasonably computable on our limited computation capacities for some datasets: few problems did not finished the extraction with (3000, 3000) for basis size. However, as it is shown in Sect. 6, small values already allow the extracted WA to be a great approximation of the RNN.

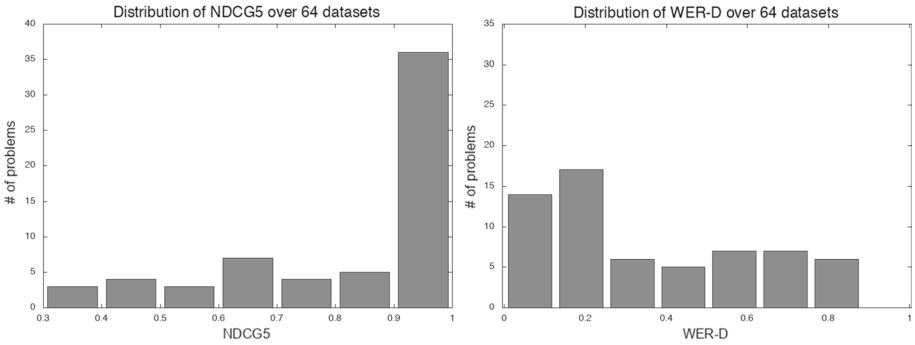


Fig. 2 Overall statistics on PAutomaC and SPiCe computed on S_{RNN}

5.4.3 Implementation details

All experiments are conducted using the Scikit-SpLearn toolbox (Arrivault et al. 2017) to handle WA and their extraction, and the PyTorch framework (van Merriënboer et al. 2017) for RNN learning and querying. A (simplified) version of our code can be find on first author's website.

6 Results

6.1 Overall quality extraction

This section globally summarizes experiments to assess the quality of extracted WA on the 62 datasets from PAutomaC and SPiCe competitions. Figure 2 shows the best obtained quality on S_{RNN} through the various hyper-parameters and sampling strategies we tested. It highlights that a large part of NDCG scores are higher than 0.9 while a majority of WER-D scores are below 0.2. This shows clearly the great ability of our algorithm to reproduce the behaviour of a LM-RNN.

In the following, we provide a finer analysis related to PAutomaC and SPiCe problems.

6.2 Basis generation

The impact of the basis for the quality of the task of WA extraction is of crucial interest. We thus carefully study the difference between the two ways to generate the basis: by sampling uniformly at random using a parameter for maximum length, and using the RNN to be distilled.

Figure 3 shows the best values obtained for each chosen metric on the evaluation datasets for the two methods. In this figure, each radius of a polar plot corresponds to one problem whose corresponding number is given outside of the circle. It details only the results on the difficult problems of the SPiCe competition: the average results on the PAutomaC problems are summed up in Table 1. The difference on these datasets is even more significant.

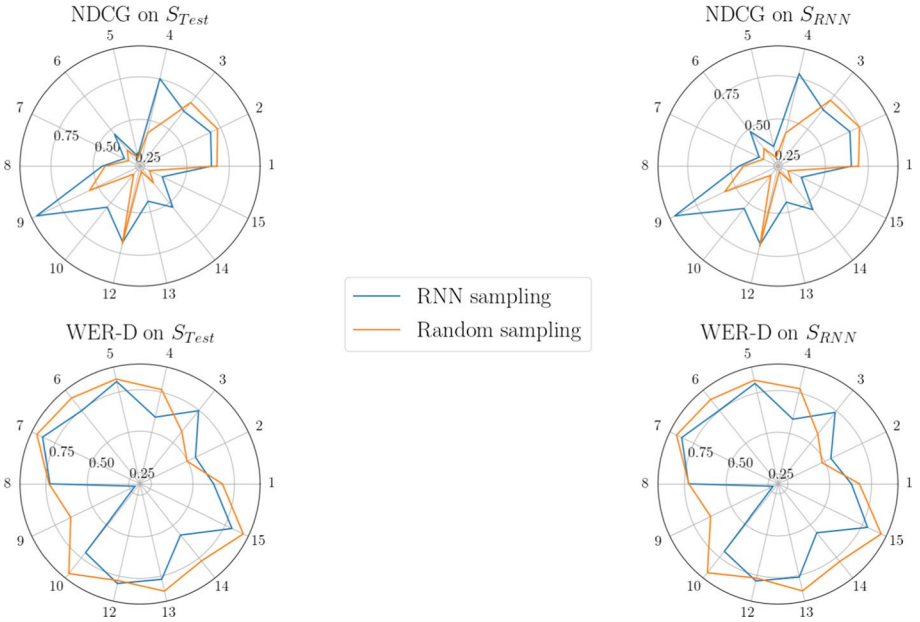


Fig. 3 Influence of the basis sampling method on the quality of the extraction for the SPiCe problems

Table 1 Comparison of the average metric values on the PAutomaC problems

	WER-D ↓		NDCG ↑	
	on S_{Test}	on S_{RNN}	on S_{Test}	on S_{RNN}
Random basis	0.334	0.336	0.851	0.848
RNN sampled basis	**0.254**	**0.259**	**0.921**	**0.915**

Recalling that best possible NDCG is 1 while best possible WER-D is 0, these elements indicate a clear trend: the generation of the prefixes and suffixes of the basis based on sampling from RNN allows better WA to be extracted.

Therefore, unless it is specified differently, all the following results concern WA extracted using a basis obtained by sampling the RNN.

6.3 Impact of the size of the basis

Figures 4 and 5 show the evolution of the NDCG and WER-D on the two evaluation sets when the size of the basis varies, on the 48 PAutomaC benchmarks. A global trend tends to occur: the larger the size of the basis is, the better the extracted WA seems to be.

This result is confirmed on SPiCe: all but one problems obtained the best quality measures with the largest basis size our experimental setup allowed us to test.

Though it is worth to be verified, this observation is not surprising: having larger basis means having access to more information about the RNN to distill, which is likely to be useful for the process.

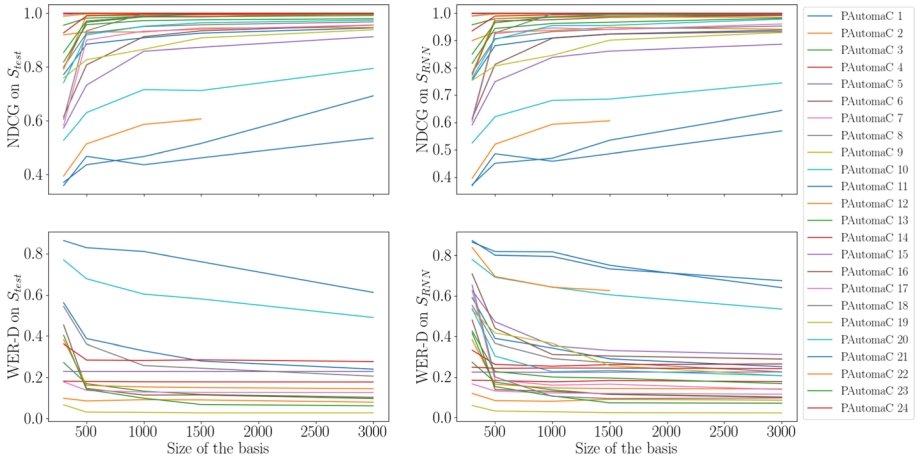


Fig. 4 Best obtained metric values on the 24th first problems of PAutomaC by the extraction using a basis sampled from the RNN. Each plot corresponds to one metric on one evaluation sample

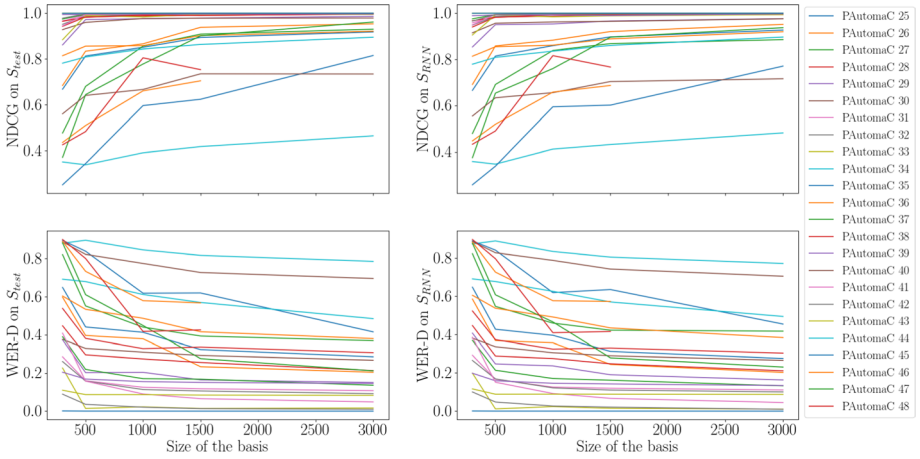


Fig. 5 Best obtained metric values on the 24th last problems of PAutomaC

It is worth noticing that, if for most PAutomaC problems we managed to run experiments up to 3000 times 3000 bases, it is not the case for SPiCe: the hardness of these tasks, in particular the large size of the vocabulary, narrowed what was possible on our (limited) computing server. Concretely, only 3 of the 14th experiments finished with 3000×3000 basis (Problems 2, 3, and 9), while two of them did not even allowed a 1500×1500 to run till the end (Problems 6 and 13).

6.4 Comparison with *n*-grams

We compared the quality of our distilled WA with a classical baseline: *n*-gram (Shannon 1948). This knowingly powerful model requires a sample of data to be learned: for each

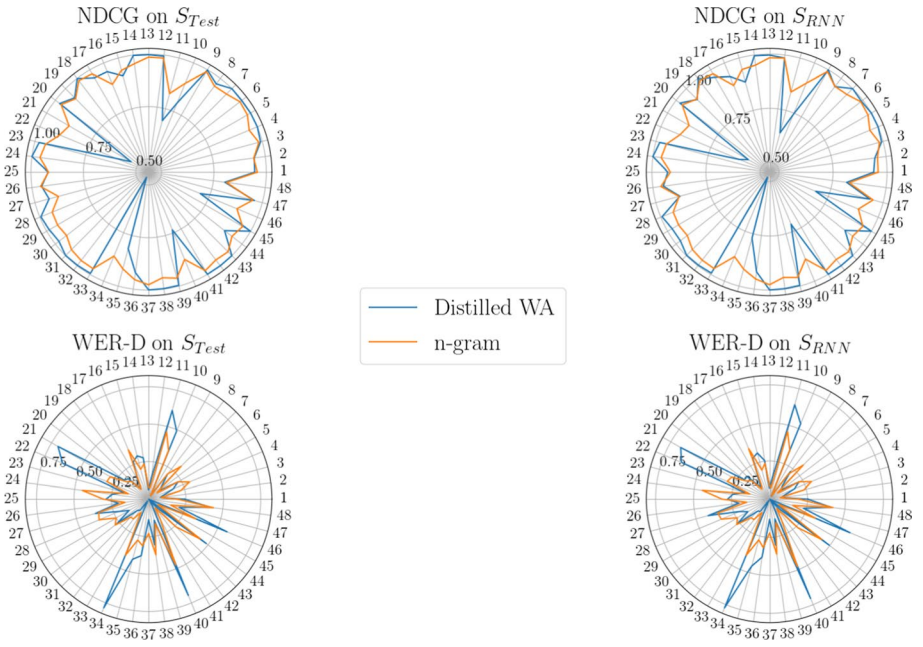


Fig. 6 Quality comparison between the best obtained results of *n*-grams and distilled WA for the PAutomaC problems

problem we randomly generate a multi-set of sequences using the corresponding trained RNN until the sum of the lengths of these strings exceeds 10 millions. This dataset is then used for training, with the window size of the model varying between 2 and 20.

Figure 6 show the comparison on the 2 evaluation sets, for the two metrics on the PAutomaC problems. Globally, the two models seems to offer comparable quality: the best distilled WA is doing slightly better than the *n*-gram on a majority of datasets, while facing some difficulties on few of them (6 over 48).

We investigated these troublesome problems to see if the characteristics of the models used to generate them can explain these lack of performance. We did not find any obvious reasons: the 3 types of machines (DPFA, PFA, HMM) are present and the values of the two sparsity parameters cover the whole range available. A common feature, though shared by problems on which the distilled WA are doing great, is that the models that generated these datasets have both a large alphabet and a big number of states. This may indicate that the Hankel matrix is not large enough, that is, it is not complete: increasing the number of prefixes and suffixes may allow better results.

Figure 7 shows the same curves for the more challenging problems of the SPiCe competition. As already noticed by Weiss et al. (2019), the *n*-gram is a (very) strong baseline on these datasets. On some problems our algorithm obtains close results but the overall difference is notable, in particular when the size of the vocabulary is important.

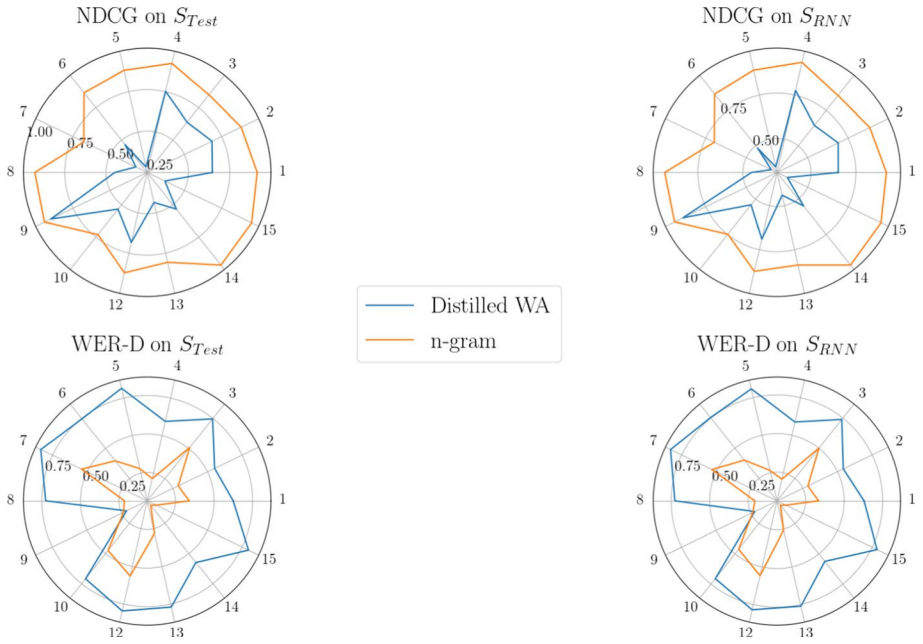


Fig. 7 Quality comparison between the best obtained results of n-grams and distilled WA for the SPiCe problems

6.5 Impact of the rank

Globally, in our experiments, the quality of the extracted WA quickly increases at first when the rank grows, then, when a relatively small rank is reached, the quality decreases and often stabilizes far from the optimum after a while. This behaviour is exemplifies on some datasets in Fig. 8 that shows the evolution of WER-D when the rank hyper-parameter increases (same plots for all the problems are given in Figs. 9 and 10 in the Appendix).

A first element of discussion is that the rank allowing the best result reinforces the interest of our distillation procedure. Indeed, the number of parameters of a WA is $|Q|^2|\Sigma| + 2|Q|$ where Q is the number of states, that is, the rank in our experiments, and Σ the alphabet: with the best overall rank average being around 175, the average number of parameters in the extracted WA is $30,625|\Sigma| + 350$ while the used RNN have already 2 million parameters just for the recurrent part (the number of parameters of the embedding layer is quadratic in the size of the alphabet and it is linear on $|\Sigma|$ for the final dense layer).

Investigating the reasons behind the relatively low-rank observation, we witness an interesting behavior of the singular values of the Hankel matrix built from the RNN. Their fast decrease, followed by a long plateau of very small values, indicates that the matrix is almost enjoying a finite number of non null singular values (the singular values for all problems are given in Figs. 11 and 12 in the Appendix). Indeed, first values are several order of magnitude larger than the following ones.

By computing the gradient of the singular values curves, and applying the same chosen threshold on them, we automatically identify for each problem a number of

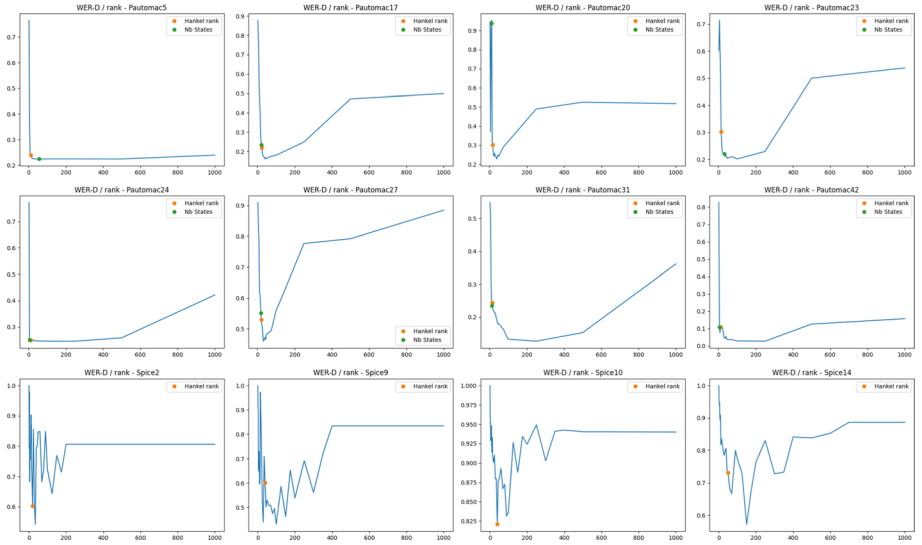


Fig. 8 Evolution of the quality in term of WER-D on S_{RNN} with the value of the rank hyper-parameter on some PAutomaC and SPiCe problems (basis size is 1000 prefixes and suffixes)

significant singular values.[4] By a slight abuse of language, we called this number the *Hankel rank*. It is shown in orange in the previously introduced figures and its numerical value.

An important observation is that the evaluation metrics at the Hankel ranks are close to the optimum. The last two columns of Tables 2 and 3 of the Appendix provide the difference of the metrics at these ranks and at the ranks allowing the best results. This shows that even when the best rank is larger than the Hankel rank—we do witness an overall increase of these best ranks when the size of the basis grows—the improvement is not significant.

This is of great interest for the understanding of the RNN behavior. Indeed, Theorem 1 states that functions whose Hankel matrix is of finite rank, that is, with a finite number of non null singular values, are exactly the ones computable by WA. Our discovery thus tends to show that our trained RNN are computing distributions almost corresponding to finite state machines, even on data that do not correspond to that kind of models, like the ones of SPiCe.

On data sets knowingly generated by finite state machines, as in PAutomaC, our study can be refined. Indeed, the green dots in the PAutomaC plot of Figs. 8, 9, and 10 , indicates the number of states of the original finite state machine. We obviously do not aim to retrieve the exact amount of states since models used for generating the PAutomaC data may require more states than WA to represent the same distribution.[5] We however notice that our framework is remarkably able to get close to the correct number of states.

[4] Though we witness small variations when the size of the bases increases, it is remarkably stable.

[5] HMM and their expressively equivalent representations of PFA may required more states than WA for the same distribution (Denis and Esposito 2008). DPFA are well-known to need more states than PFA to represent the same stochastic language.

Table 2 Detailed results on the PAutomaC benchmark

Pb	NDCG ↑				WER-D ↓				RNN		Δ_{NDCG}	$\Delta_{\text{WER-D}}$
	S_{Test}		S_{RNN}		S_{Test}		S_{RNN}					
	WA	n-gram	WA	n-gram	WA	n-gram	WA	n-gram	loss	HR		
1	0.945 (3000, 100)	0.9569 (4)	0.938 (3000, 100)	0.9587 (4)	0.24 (3000, 250)	0.1957 (4)	0.252 (3000, 100)	0.1952 (4)	0.008	15	0.063	0.181
2	0.947 (3000, 1000)	0.9444 (3)	0.948 (1500, 250)	0.9602 (3)	0.079 (3000, 1000)	0.0837 (3)	0.081 (1000, 24)	0.0695 (3)	0.007	13	0.021	0.017
3	0.991 (3000, 250)	0.9832 (6)	0.993 (3000, 250)	0.9824 (6)	0.103 (3000, 500)	0.2236 (6)	0.101 (3000, 250)	0.2343 (6)	0.006	18	0.018	0.092
4	1.0 (500, 100)	0.9579 (5)	1.0 (500, 100)	0.9584 (5)	0.178 (3000, 1000)	0.297 (5)	0.177 (1000, 100)	0.2987 (5)	0.005	13	0.001	0.003
5	1.0 (100, 39)	0.9827 (4)	1.0 (100, 42)	0.9824 (4)	0.229 (3000, 33)	0.2281 (9)	0.224 (1000, 18)	0.2255 (8)	0.003	10	0.001	0.015
6	0.998 (3000, 100)	0.9879 (6)	0.992 (3000, 250)	0.9882 (6)	0.097 (3000, 250)	0.0698 (6)	0.102 (3000, 250)	0.066 (6)	0.004	16	0.014	0.021
7	1.0 (500, 75)	0.9637 (3)	1.0 (1000, 250)	0.9622 (3)	0.126 (3000, 1000)	0.3078 (4)	0.116 (3000, 1000)	0.314 (4)	0.006	16	0.004	0.052
8	0.968 (3000, 250)	0.9543 (4)	0.953 (3000, 250)	0.9537 (4)	0.206 (3000, 250)	0.2106 (4)	0.226 (3000, 250)	0.2087 (4)	0.007	28	0.047	0.112
9	0.998 (1500, 250)	0.9899 (6)	1.0 (3000, 500)	0.9908 (6)	0.027 (1500, 250)	0.0261 (6)	0.024 (3000, 500)	0.0262 (6)	0.002	16	0.012	0.036
10	0.794 (3000, 100)	0.8955 (4)	0.744 (3000, 75)	0.899 (4)	0.49 (3000, 100)	0.3483 (4)	0.535 (3000, 75)	0.3479 (4)	0.007	20	0.114	0.15
11	0.692 (3000, 75)	0.8254 (3)	0.643 (3000, 75)	0.8381 (3)	0.612 (3000, 75)	0.4654 (3)	0.641 (3000, 100)	0.4586 (3)	0.008	21	0.178	0.221
12	0.995 (3000, 15)	0.9855 (4)	0.986 (1500, 48)	0.9852 (4)	0.145 (3000, 100)	0.0756 (3)	0.142 (3000, 75)	0.0757 (3)	0.005	15	0.011	0.025
13	0.997 (1500, 250)	0.9853 (8)	0.995 (1500, 250)	0.983 (7)	0.062 (3000, 500)	0.0791 (7)	0.072 (3000, 250)	0.0882 (7)	0.004	30	0.045	0.144
14	0.998 (3000, 75)	0.9583 (3)	0.997 (1500, 250)	0.9533 (3)	0.277 (3000, 500)	0.2371 (4)	0.255 (1000, 100)	0.2359 (4)	0.008	12	0.065	0.062
15	0.912 (3000, 45)	0.9393 (3)	0.886 (3000, 42)	0.9453 (3)	0.297 (3000, 42)	0.2044 (3)	0.312 (3000, 42)	0.1958 (3)	0.007	20	0.035	0.057
16	0.955 (3000, 75)	0.8905 (4)	0.932 (3000, 100)	0.8881 (4)	0.268 (3000, 250)	0.3546 (4)	0.29 (3000, 100)	0.3568 (4)	0.007	19	0.225	0.288
17	0.956 (3000, 100)	0.978 (3)	0.96 (3000, 75)	0.9794 (3)	0.146 (3000, 100)	0.1251 (3)	0.14 (3000, 75)	0.1108 (3)	0.006	22	0.021	0.049
18	0.999 (3000, 250)	0.9866 (4)	0.997 (1000, 75)	0.9868 (4)	0.084 (3000, 250)	0.0841 (3)	0.095 (1500, 42)	0.0823 (3)	0.005	23	0.107	0.0
19	0.938 (3000, 250)	0.9318 (5)	0.929 (3000, 250)	0.9314 (5)	0.19 (3000, 250)	0.1639 (5)	0.207 (3000, 250)	0.1697 (5)	0.005	12	0.201	0.35
20	0.974 (3000, 42)	0.9656 (3)	0.978 (3000, 39)	0.965 (3)	0.208 (3000, 42)	0.1902 (3)	0.208 (3000, 39)	0.1936 (3)	0.008	15	0.022	0.019
21	0.535 (3000, 100)	0.8772 (3)	0.569 (3000, 100)	0.8853 (3)	0.701 (3000, 100)	0.2965 (3)	0.675 (3000, 100)	0.2875 (3)	0.008	22	0.267	0.267
22	0.606 (1500, 75)	0.8985 (3)	0.606 (1500, 75)	0.9043 (3)	0.625 (1500, 39)	0.3029 (3)	0.627 (1500, 75)	0.3003 (3)	0.007	27	0.028	0.018
23	0.979 (3000, 250)	0.9475 (4)	0.982 (3000, 250)	0.9501 (4)	0.174 (3000, 250)	0.1409 (4)	0.169 (3000, 250)	0.1341 (4)	0.006	12	0.067	0.123
24	1.0 (100, 18)	0.9613 (5)	1.0 (100, 18)	0.9615 (5)	0.246 (500, 100)	0.4483 (3)	0.243 (3000, 39)	0.4468 (5)	0.005	10	0.001	0.004

Table 2 (continued)

Pb	NDCG ↑				WER-D ↓				RNN		Δ_{NDCG}	$\Delta_{\text{WER-D}}$
	S_{Test}		S_{RNN}		S_{Test}		S_{RNN}					
	WA	*n*-gram	WA	*n*-gram	WA	*n*-gram	WA	*n*-gram	loss	HR		
25	0.918 (3000, 250)	0.9163 (4)	0.927 (3000, 250)	0.9146 (4)	0.285 (3000, 250)	0.3035 (4)	0.276 (3000, 250)	0.3113 (4)	0.008	9	0.177	0.252
26	0.953 (3000, 100)	0.9587 (5)	0.951 (3000, 100)	0.9629 (5)	0.204 (3000, 250)	0.1592 (6)	0.203 (3000, 250)	0.1509 (6)	0.005	34	0.073	0.2
27	0.93 (3000, 45)	0.885 (3)	0.885 (3000, 45)	0.8891 (3)	0.369 (3000, 100)	0.3462 (3)	0.418 (3000, 45)	0.3527 (3)	0.008	17	0.036	0.083
28	0.995 (3000, 500)	0.9372 (4)	0.994 (3000, 500)	0.9383 (4)	0.213 (3000, 500)	0.3591 (4)	0.212 (3000, 500)	0.3431 (4)	0.008	10	0.054	0.351
29	0.986 (3000, 100)	0.9413 (5)	0.976 (3000, 250)	0.9418 (5)	0.15 (3000, 500)	0.2516 (5)	0.164 (3000, 1000)	0.244 (5)	0.005	20	0.042	0.144
30	0.979 (3000, 33)	0.9329 (3)	0.976 (3000, 250)	0.9353 (3)	0.267 (3000, 500)	0.2819 (3)	0.266 (3000, 250)	0.2713 (3)	0.008	11	0.018	0.07
31	0.999 (3000, 500)	0.9599 (4)	0.998 (3000, 250)	0.9604 (4)	0.108 (3000, 500)	0.1698 (5)	0.111 (3000, 250)	0.1726 (5)	0.006	12	0.013	0.123
32	0.997 (3000, 1000)	0.9684 (7)	0.996 (3000, 500)	0.9673 (7)	0.092 (3000, 1000)	0.1659 (8)	0.101 (3000, 500)	0.1685 (7)	0.005	24	0.016	0.097
33	0.995 (1500, 39)	0.969 (3)	0.994 (3000, 42)	0.969 (3)	0.014 (500, 6)	0.1008 (3)	0.01 (3000, 50)	0.0999 (3)	0.007	11	0.02	0.04
34	0.465 (3000, 50)	0.8707 (3)	0.482 (3000, 50)	0.8775 (3)	0.785 (3000, 50)	0.4048 (3)	0.773 (3000, 50)	0.398 (3)	0.008	15	0.07	0.076
35	0.815 (3000, 100)	0.9148 (3)	0.771 (3000, 75)	0.9186 (3)	0.415 (3000, 100)	0.2816 (3)	0.455 (3000, 100)	0.2757 (3)	0.006	26	0.23	0.284
36	0.921 (3000, 50)	0.9542 (3)	0.919 (3000, 50)	0.9527 (3)	0.38 (3000, 100)	0.3128 (3)	0.384 (3000, 48)	0.3212 (3)	0.008	13	0.089	0.234
37	0.999 (3000, 250)	0.9747 (3)	0.999 (3000, 250)	0.9752 (3)	0.136 (3000, 250)	0.2277 (3)	0.133 (3000, 50)	0.2206 (3)	0.009	11	0.008	0.145
38	0.998 (3000, 250)	0.9455 (3)	0.998 (3000, 250)	0.9459 (3)	0.307 (3000, 250)	0.3744 (3)	0.303 (3000, 250)	0.3643 (3)	0.009	6	0.017	0.123
39	0.996 (3000, 30)	0.9608 (4)	0.997 (3000, 100)	0.9588 (4)	0.145 (3000, 100)	0.1687 (4)	0.134 (3000, 6)	0.1583 (4)	0.006	9	0.006	0.135
40	0.735 (1500, 48)	0.888 (3)	0.716 (3000, 50)	0.8898 (3)	0.695 (3000, 100)	0.4751 (3)	0.706 (3000, 100)	0.4795 (3)	0.009	10	0.129	0.211
41	1.0 (3000, 250)	0.9793 (3)	1.0 (3000, 250)	0.9792 (3)	0.048 (3000, 250)	0.1602 (3)	0.045 (3000, 250)	0.1579 (3)	0.008	7	0.045	0.432
42	1.0 (1500, 250)	0.9648 (4)	0.999 (3000, 500)	0.9632 (4)	0.01 (3000, 250)	0.0723 (4)	0.01 (3000, 100)	0.0761 (4)	0.006	9	0.01	0.089
43	1.0 (100, 45)	0.9743 (4)	1.0 (100, 45)	0.9743 (4)	0.083 (3000, 500)	0.2612 (2)	0.088 (3000, 12)	0.2649 (2)	0.008	8	0.001	0.041
44	0.895 (3000, 50)	0.9396 (3)	0.896 (3000, 50)	0.9408 (2)	0.485 (3000, 250)	0.435 (3)	0.495 (3000, 50)	0.434 (3)	0.009	8	0.085	0.195
45	1.0 (500, 45)	0.956 (3)	1.0 (100, 30)	0.9548 (3)	0.0 (500, 27)	0.0732 (2)	0.0 (500, 75)	0.0708 (2)	0.009	4	0.001	0.001
46	0.705 (1500, 45)	0.8945 (3)	0.687 (1500, 39)	0.9011 (3)	0.567 (1500, 45)	0.3421 (3)	0.573 (1500, 42)	0.3258 (3)	0.009	10	0.154	0.231
47	0.961 (3000, 100)	0.9536 (4)	0.937 (3000, 100)	0.952 (4)	0.212 (3000, 100)	0.1531 (4)	0.231 (3000, 100)	0.1571 (4)	0.005	9	0.52	0.593

Table 2 (continued)

Pb	NDCG ↑				WER-D ↓				RNN		Δ_{NDCG}	$\Delta_{\text{WER-D}}$
	S_{Test}		S_{RNN}		S_{Test}		S_{RNN}					
	WA	*n*-gram	WA	*n*-gram	WA	*n*-gram	WA	*n*-gram	loss	HR		
48	0.805 (1000, 24)	0.8139 (3)	0.816 (1000, 24)	0.8282 (3)	0.417 (1000, 39)	0.4369 (3)	0.41 (1000, 24)	0.4209 (3)	0.008	13	0.108	0.231

Table 3 Detailed results on the SPiCe benchmark

Pb	NDCG ↑				WER-D ↓				RNN			
	S_{Test}		S_{RNN}		S_{Test}		S_{RNN}					
	WA	n-gram	WA	n-gram	WA	n-gram	WA	n-gram	loss	HR	Δ_{NDCG}	$\Delta_{\text{WER-D}}$
1	0.645 (1000, 175)	0.9569 (4)	0.652 (100, 10)	0.9587 (4)	0.624 (3000, 40)	0.1957 (4)	0.629 (1500, 60)	0.1952 (4)	0.003	14	0.191	0.068
2	0.686 (1500, 30)	0.9444 (3)	0.689 (1500, 30)	0.9602 (3)	0.552 (1000, 35)	0.0837 (3)	0.542 (1000, 35)	0.0695 (3)	0.002	17	0.112	0.061
3	0.639 (3000, 90)	0.9832 (6)	0.645 (3000, 90)	0.9824 (6)	0.746 (3000, 90)	0.2236 (6)	0.739 (3000, 500)	0.2343 (6)	0.002	19	0.078	0.149
4	0.752 (1500, 250)	0.9579 (5)	0.776 (1500, 250)	0.9584 (5)	0.595 (1000, 250)	0.297 (5)	0.588 (1500, 250)	0.2987 (5)	0.005	165	0.021	0.014
5	0.284 (500, 20)	0.9827 (4)	0.342 (1000, 2)	0.9824 (4)	0.815 (500, 2)	0.2281 (9)	0.808 (500, 2)	0.2255 (8)	0.001	15	0.05	0.039
6	0.466 (1000, 35)	0.9879 (6)	0.483 (1000, 50)	0.9882 (6)	0.747 (1000, 60)	0.0698 (6)	0.754 (1000, 50)	0.066 (6)	0.002	31	0.016	0.036
7	0.326 (3000, 150)	0.9637 (3)	0.346 (3000, 150)	0.9622 (3)	0.833 (500, 3)	0.3078 (4)	0.83 (500, 3)	0.314 (4)	0.002	4	0.115	0.009
8	0.446 (1500, 25)	0.9543 (4)	0.452 (1500, 25)	0.9537 (4)	0.725 (100, 4)	0.2106 (4)	0.726 (100, 4)	0.2087 (4)	0.002	28	0.069	0.164
9	0.9 (3000, 50)	0.9899 (6)	0.891 (3000, 150)	0.9908 (6)	0.215 (3000, 200)	0.0261 (6)	0.223 (3000, 150)	0.0262 (6)	0.001	36	0.133	0.26
10	0.534 (1500, 40)	0.8955 (4)	0.541 (1500, 40)	0.899 (4)	0.709 (1500, 40)	0.3483 (4)	0.705 (1500, 40)	0.3479 (4)	0.003	33	0.0	0.0
12	0.683 (1500, 70)	0.8254 (3)	0.69 (1500, 70)	0.8381 (3)	0.796 (1500, 70)	0.4654 (3)	0.785 (1500, 70)	0.4586 (3)	0.002	33	0.161	0.141
13	0.436 (500, 90)	0.9855 (4)	0.44 (500, 90)	0.9852 (4)	0.77 (500, 90)	0.0756 (3)	0.761 (500, 90)	0.0757 (3)	0.004	22	0.235	0.171
14	0.534 (1000, 150)	0.9853 (8)	0.549 (1500, 150)	0.983 (7)	0.574 (1000, 150)	0.0791 (7)	0.562 (1500, 150)	0.0882 (7)	0.001	46	0.229	0.197
15	0.369 (1000, 60)	0.9583 (3)	0.375 (1000, 60)	0.9533 (3)	0.796 (1000, 60)	0.2371 (4)	0.785 (1000, 60)	0.2359 (4)	0.002	45	0.116	0.127

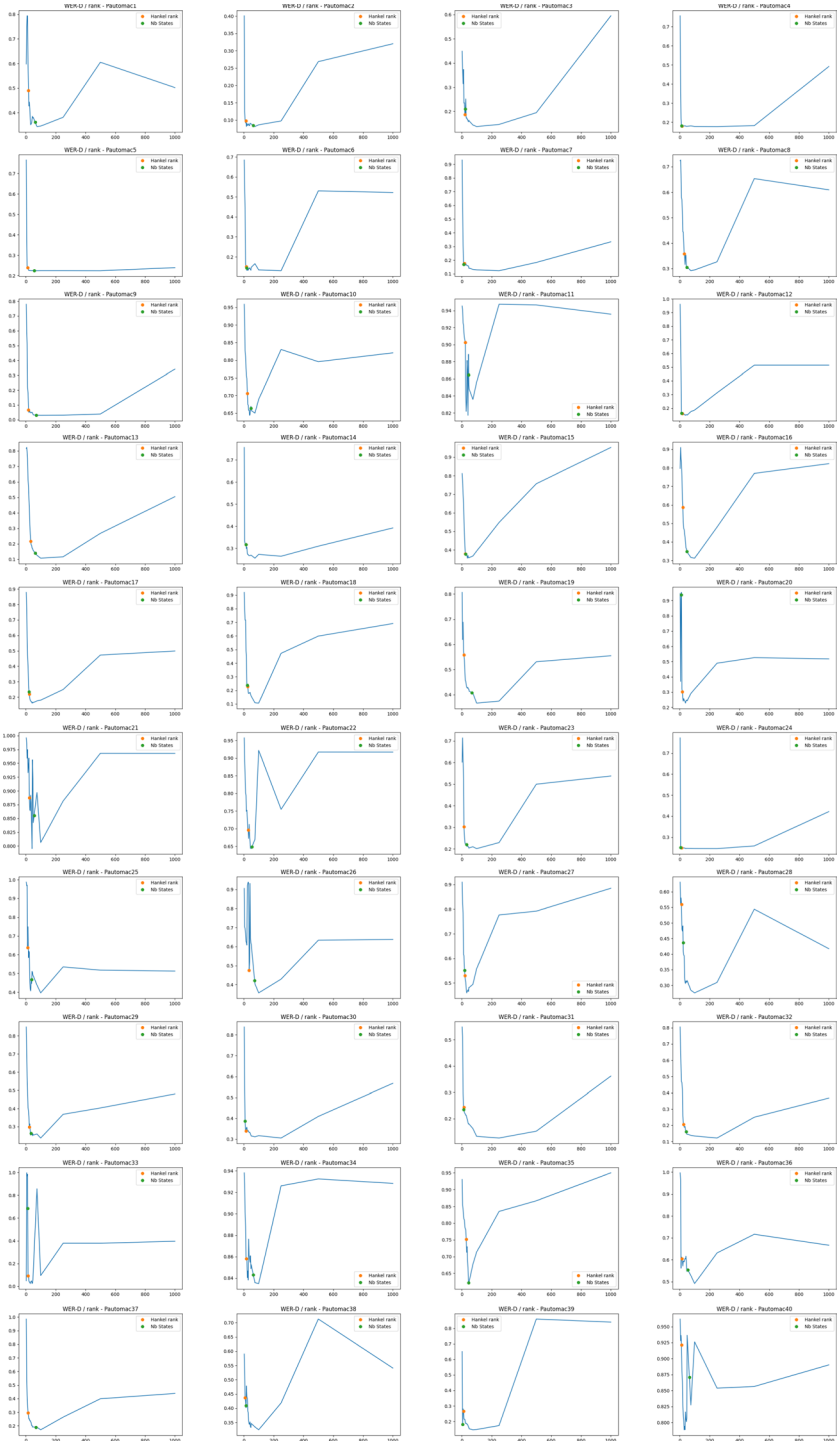


Fig. 9 Evolution of the quality in term of WER-D with the value of the rank hyper-parameter on the 40 first PAutomaC problems, evaluated on S_{RNN} with basis size 1000×1000

Fig. 10 Evolution of the quality in term of WER-D with the value of the rank hyper-parameter on the 8 last PAutomaC problems and the SPiCe ones, evaluated on S_{RNN} with basis size 1000×1000

This provides important understandings both on the quality of the learned RNN—as the number of significant singular values is close to the number of states of the target, validating the quality of the learning phase—and of our distillation process—as the quality of the extracted WA with Hankel rank states is almost always comparable to the optimum.

Finally, as stated in different theoretical works summarized in Sect. 2, finite precision and time GRU RNN are expected to behave like finite state machines (at least asymptotically), and our empirical conclusions tend to validate this point. This is why we re-run our experiments using LSTM cells instead of GRU ones: we observed the exact same

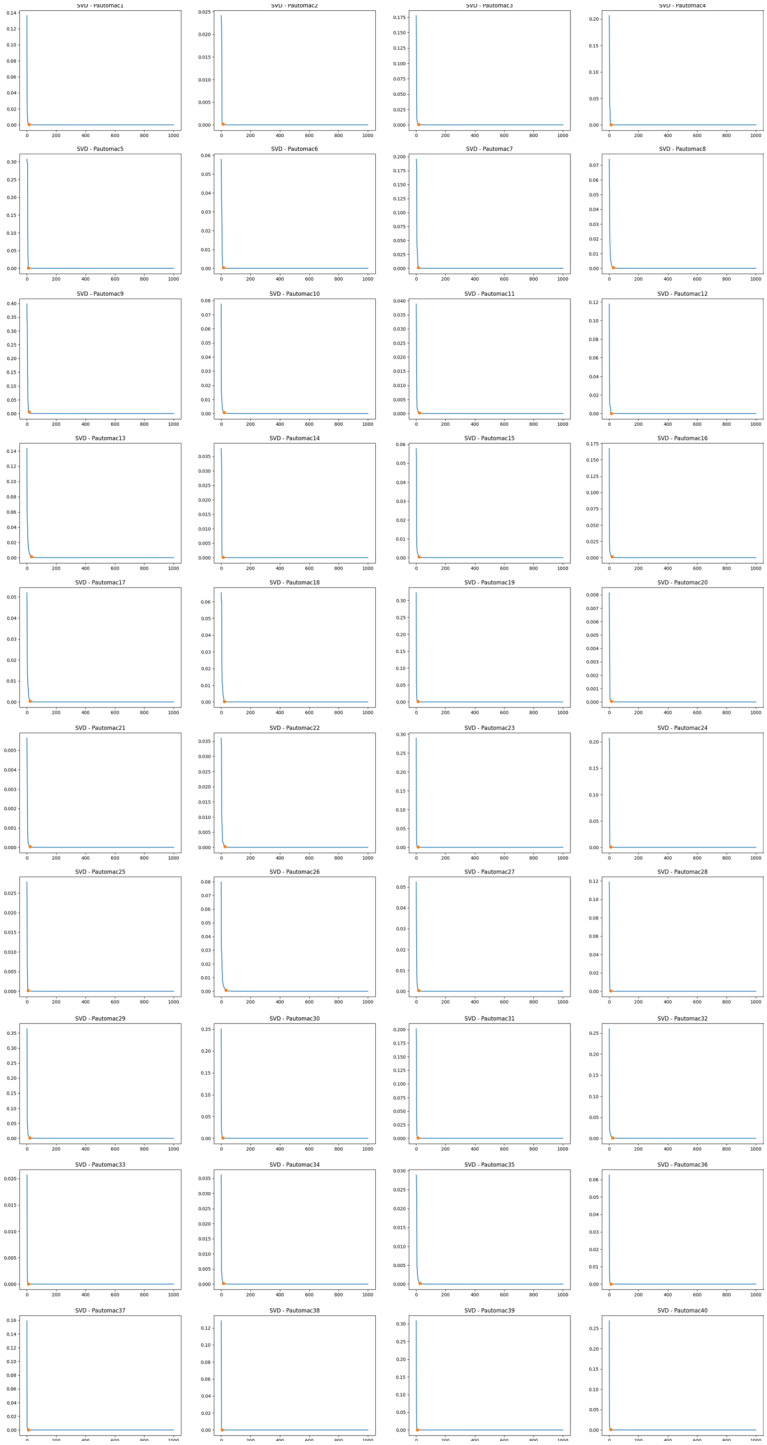


Fig. 11 Singular values of the Hankel matrix of the RNN learned on the 40 first PAutomaC datasets, with a 1000×1000 basis. So called *Hankel rank* is shown by a orange dot

Fig. 12 Singular values of the Hankel matrix of the RNN learned on the last 8 PAutomaC datasets and the SPiCe ones, with 1000×1000 bases

behaviour despite the fact that the theoretical studies indicates a greater expressivity power for this type of RNN. We think this finding is an important outcome of our study.

7 Conclusion

This article presents a spectral approach to extract weighted automata from recurrent neural networks already trained on a language modelling task. The proposed algorithm uses the RNN as an oracle, querying it to know values it assigned to carefully selected data, and then produces the automaton using a singular value decomposition of the

Hankel matrix, a mathematical object it uses to store the RNN answers. Importantly, the algorithm does not require access to the inner representation of the networks: this allows it to work on any type of black box that computes a real value from an input sequence—like models used for sentiment analysis from text (Chaudhuri 2019)—or that can be straightforwardly used for this task—like language modelling RNN. This point is an important advantage of the proposed method compared to existing ones.

Our experiments on 62 different datasets show the overall quality of the extraction in the sense that the obtained WA represent good approximations of the distributions of the RNN. This validates our distillation process: WA computing linear functions, our algorithm provides linear, and thus computably efficient, models from non linear ones. This is emphasized by the fact that even the small extracted WA are already acceptable approximations, while requiring far less parameters.

Moreover, our process allows a deeper understanding of the underlying decision process of the trained RNN: the quick decrease of the singular values of the Hankel matrices indicates that these RNN are computing (approximations of) rational series, that is, their behaviour is the one of a linear finite state machine. This peek into the internal and opaque functioning of RNN allows to raise an fundamental question : to which extend RNN trained on data are biased towards functions of the lowest level of the Chomsky hierarchy?

Though interpretability of automata is well-established, partly but not entirely due to the existence of a graphical representation (see Hammerschmidt et al. (2016) for a detailed discussion), one can argue that the non-deterministic nature of our WA, and the potential presence of negative weights, diminish the interest of the proposed work toward explainability. If this critic is understandable, the observation of the quick decrease of the singular values reinforces the merit of our approach for that task. Moreover, WA enjoy characteristics that RNN do not, as for instance the existence of a computable distance between WA (Droste et al. 2009), the opportunity to efficiently decide their equivalence (Tzeng 1992), or the possibility to modify any WA such that it consists of a deterministic tree-structured part as deep as one wants followed by a non-deterministic part (Bailly 2011). These elements indicate clear paths for the continuation of the study presented here.

This paper rises other questions that constitute potential interesting future works. The first one is to take into account the observation on the size of the singular values directly in a distillation algorithm. For instance, one can imaging that another, smaller RNN can be designed from this information, following the framework of born again neural networks (Furlanello et al. 2018).

Another interesting line of research is to extend this approach to multi-valued weighted automata (Rabusseau et al. 2017): these finite state machines can directly be trained on a language modelling task or can be used for multi-task learning. Testing whether this allows better distillation is an important question that should be answered.

Extending our study to other types of RNN is also something that is worth to be mined. In particular, it would be interesting to know if second order RNN learned on data exhibit the same behavior than the one observed here: as their linear version is equivalent to WA (Rabusseau et al. 2019), the process would correspond to a linearization.

Verifying whether bi-directional RNN (Schuster et al. 1997) also share the finite state flavour that we detect in this paper provides also an interesting future line of research. Because of the works on learning context-free grammars (Clark and Eyraud 2007; Clark et al. 2010) and beyond context-free (Clark and Yoshinaka 2014), we conjecture that it would not be the case. Indeed, bi-directional RNN use information from both the prefix

and the suffix of a given element, which positions them within the spectrum of distributional learning (Clark and Yoshinaka 2016). The work on spectral learning of non finite state models (Bailly et al. 2013) would then be then a good starting point for a distillation process for these networks.

Appendix

Tables 2 and 3 gives numerical detailed results of our experiments on PAutomaC and SpiCe, respectively. For each metric, for each evaluation set, the tables give the best value obtained by the spectral extraction with a RNN sampled basis and the n-gram baseline. For the spectral extraction, corresponding best parameters are given as a pair: the first value is the number of prefixes and suffixes of the basis, the second one is the rank. The value between parenthesis for n-gram is the size of the window. To give an idea of the quality of the trained RNN on each problem, the tables also provide their loss on a validation set. The Table also gives the Hankel rank for each RNN for bases made of 1000 prefixes and suffixes (column *HR*). The difference between the best obtained NDCG [resp. WER-D] by a WA and the NDCG [resp. WER-D] at the Hankel rank for the corresponding basis size is given in column Δ_{NDCG} [resp. $\Delta_{\text{WER-D}}$].

Acknowledgements We want to thanks our former student Noé Goudian for developing the first version of the code used for the experiments. We owe likewise a great debt of gratitude to Gail Weiss for the long discussions, the code sharing, and the hours passed confronting our results. We are also very grateful to Guillaume Rabusseau, François Denis, and Matthias Galé for fruitful discussions and for pointing to us important pieces of literature. This work was performed using HPC resources from Centre de Calcul Intensif dAix-Marseille and GENCI-IDRIS (Grant 2020-AD011011920).

References

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75*(2), 87–106.

Arrivault, D., Benielli, D., Denis, F., & Eyraud, R. (2017). Scikit-SpLearn: A toolbox for the spectral learning of weighted automata compatible with scikit-learn. *Conférence francophone en*. (**Apprentissage**).

Avcu, E., Shibata, C.,& Heinz, J. (2017). Subregular complexity and deep learning. In *Proceedings of the Conference on Logic and Machine Learning in Natural Language*, pp. 20–33.

Ayache, S., Eyraud, R.,& Goudian, N. (2018). Explaining black boxes on sequential data using weighted automata. In *Proceedings of the International Conference on Grammatical Inference*, Vol 93 of *PMLR*, pp. 81–103.

Bailly, R. (2011). *Méthodes spectrales pour l'inférence grammaticale probabiliste de langages stochastiques rationnels*. PhD thesis, Aix-Marseille University.

Bailly, R., Denis, F., & Ralaivola, L. (2009). Grammatical inference as a principal component analysis problem. In*International Conference on Machine Learning,* pp. 33–40.

Bailly, R., Carreras, X., Luque, F. M., & Quattoni, A. (2013). Unsupervised spectral learning of WCFG as low-rank matrix completion. In: *Proceedings of EMNLP,* pp. 624–635.

Balle, B.,& Mohri, M. (2015). Learning weighted automata. In *Algebraic Informatics - 6th International conference, CAI 2015, proceedings*, pages 1–21.

Balle, B., Carreras, X., Luque, F., & Quattoni, A. (2014). Spectral learning of weighted automata. *Machine Learning, 96*(1–2), 33–63.

Balle, B., Eyraud, R., Luque, F. M., Quattoni, A.,& Verwer, S. (2017). Results of the sequence prediction challenge (SPiCe): a competition on learning the next symbol in a sequence. In *Proceedings of the international conference on grammatical inference*, Vol. 57 of *PMLR*, pp. 132–136.

Berstel, J., & Reutenauer, C. (1988). *Rational Series and Their Languages* (p. 0387186263). Berlin, Heidelberg: Springer-Verlag.
Carlyle, J. W., & Paz, A. (1971). Realizations by stochastic finite automata. *Journal of Computer and System Sciences, 5*(1), 26–40.
Cechin, A. L., Regina, D., Simon, P.,& Stertz, K. (2003). State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *Proceedings of the International Conference of the Chilean Computer Science Society*, pp. 73–78.
Chaudhuri, A. (2019). *Visual and Text Sentiment Analysis through Hierarchical Deep Learning Networks* (1st ed., p. 9789811374739). Incorporated: Springer Publishing Company.
Chen, Y., Gilroy, S., Knight, K.,& May, J. (2017). Recurrent neural networks as weighted language recognizers. *CoRR*, arXiv:abs/1711.05408.
Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H.,& Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, arXiv:abs/1406.1078.
Chung, H., Iorga, M., Voas, J., & Lee, S. (2017). Alexa, can i trust you? *IEEE Computer, 50*(9), 100–104.
Clark, A., & Eyraud, R. (2007). Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research, 8,* 1725.
Clark, A., & Yoshinaka, R. (2014). Distributional learning of parallel multiple context-free grammars. *Machine Learning, 96*(1–2), 5–31.
Clark, A., & Yoshinaka, R. (2016). *Distributional Learning of Context-Free and Multiple Context-Free Grammars* (pp. 143–172). Berlin Heidelberg: Springer.
Clark, A., Eyraud, R., & Habrard, A. (2010). Using contextual representations to efficiently learn context-free languages. *Journal of Machine Learning Research, 11,* 2707–2744.
Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation, 1*(3), 372–381.
Craven, M. W., & Shavlik, J. W. (1995). Extracting tree-structured representations of trained networks. In *Proceedings of the international conference on neural information processing systems*, pp. 24–30.
Denis, F., & Esposito, Y. (2008). On rational stochastic languages. *Fundamenta Informaticae, 86*(1–2), 41–77.
Deoras, A., Mikolov, T., Kombrink, S., Karafiát, M., & Khudanpur, S. (2011). Variational approximation of long-span language models for lvcsr. In *Proceedings of the international conference on acoustics, speech and signal processing*, pp. 5532–5535.
Dimitrova, E. S., Vera Licona, M. P., McGee, J., & Laubenbacher, R. (2010). Discretization of time series data. *Journal of Computational Biology, 17*(6), 853–868.
Droste, M., Kuich, W., & Vogler, H. (2009). *Handbook of Weighted Automata* (1st ed.). Berlin: Springer Publishing Company.
Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*(2), 179–211.
Flies, M. (1974). Matrice de hankel. *Journal de Mathématique Pures et Appliquées, 5,* 197–222.
Frosst, N., & Hinton, G. E. (2017). Distilling a neural network into a soft decision tree. In *Proceedings of the first international workshop on comprehensibility and explanation in AI and ML*.
Furlanello, T., Lipton, Z., Tschannen, M., Itti, L.,& Anandkumar, A. (2018). Born again neural networks. In *Proceedings of the international conference on machine learning*, Vol. 80 of *PMLR*, pp. 1607–1616.
Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., & Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation, 4*(3), 393–405.
Graves, A., Mohamed, A.,& Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649.
Guidotti, R., Monreale, A., Turini, F., Pedreschi, D.,& Giannotti, F. (2018). A survey of methods for explaining black box models. *CoRR*, arXiv:abs/1802.01933.
Hammerschmidt, C. A., Verwer, S., Lin, Q.,& State, R. (2016). Interpreting finite automata for sequential data. *CoRR*, arXiv:abs/1611.07100.
Harmon, R. R.,& Auseklis, N. (2009). Sustainable it services: Assessing the impact of green computing practices. In *PICMET '09 - 2009 Portland international conference on management of engineering technology*, pp. 1707–1717.
Hayashi, Y., & Nakai, M. (1990). Automated extraction of fuzzy if-then rules using neural networks. *IEEJ Transactions on Electronics, Information and Systems, 110*(3), 198–206.
Hinton, G., Vinyals, O.,& Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS deep learning and representation learning workshop*.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Boston: Addison-Wesley Publishing Company.

Hsu, D., Kakade, S.,& Zhang, T. (2009). A spectral algorithm for learning hidden markov models. In *Conference on computational learning theory*.

Jacobsson, H. (2005). Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation, 17*(6), 1223–1263.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C. Shannon & J. McCarthy (Eds.), *Automata Studies* (pp. 3–41). Princeton, NJ: Princeton University Press.

Lecorvé, G.,& Motlícek, P. (2012). Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition. In *Proceedings of the conference of the international speech communication association*, pp. 1668–1671.

Li, J., Monroe, W.,& Jurafsky, D. (2016). Understanding neural networks through representation erasure. *CoRR*, arXiv:abs/1612.08220.

Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, arXiv:abs/1506.00019.

Mahalunkar, A., & Kelleher, J. D. (2019). Multi-element long distance dependencies: Using SPk languages to explore the characteristics of long-distance dependencies. *CoRR*, arXiv:abs/1907.06048.

Manolios, P., & Fanelli, R. (1994). First-order recurrent neural networks and deterministic finite state automata. *Neural Computation, 6*(6), 1155–1173.

Marzouk, R., & de la Higuera, C. (2020). Distance and equivalence between finite state machines and recurrent neural networks: Computational results. *CORR*. https://arxiv.org/abs/2004.00478.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics, 5*(4), 115–133.

Merrill, W. (2019). Sequential neural networks as automata. *CoRR*, arXiv:abs/1906.01615.

Merrill, W., Weiss, G., Goldberg, Schwartz, R., Smith, N. A., & Yahav, E. (2020). A formal hierarchy of rnn architectures. *ArXiv*, arXiv:abs/2004.08500.

Metropolis, N., & Ulam, S. (1949). The monte carlo method. *Journal American Statistical Association, 44,* 335.

Mohri, M. (2009). Weighted automata algorithms. In M. Droste, W. Kuich, & H. Vogler (Eds.), *Handbook of Weighted Automata* (pp. 213–254). Berlin Heidelberg: Springer.

Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society, 26,* 394–395.

Mozer, M. C., Kazakov, D., & Lindsey, R. V. (2018). State-denoised recurrent neural networks. *CoRR*, arXiv:abs/1805.08394.

Okudono, T., Waga, M., Sekiyama, T., & Hasuo, I. (2019). Weighted automata extraction from recurrent neural networks via regression on state spaces. *CoRR*, arXiv:abs/1904.02931.

Omlin, C. W., & Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks, 9*(1), 41–52.

Quattoni, A., Carreras, X.,& Gallé, M. (2017). A maximum matching algorithm for basis selection in spectral learning. In *Proceedings of the 20th international conference on artificial intelligence and statistics*, Vol. 54 of *PMLR*, pp 1477–1485.

Rabusseau, G., Balle, B., & Pineau, J. (2017). Multitask spectral learning of weighted automata. In *Advances in neural information processing systems,* pp. 2585–2594.

Rabusseau, G., Li, T.,& Precup, D. (2019). Connecting weighted automata and recurrent neural networks through spectral learning. In *Proceedings of AISTATS*, Vol. 89 of *PMLR*, pp. 1630–1639.

Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior, 6,* 855–863.

Túlio Ribeiro, M., Singh, S.,& Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the international conference on knowledge discovery and data mining*, pp. 1135–1144.

Rubinstein, R. Y., & Kroese, D. P. (2004). *The cross entropy method: a unified approach to combinatorial optimization. Monte–Carlo simulation* (p. 038721240X). Berlin, Heidelberg: Springer-Verlag.

Rudin, W. (1976). *Principles of mathematical analysis. International series in pure and applied mathematics*. New York: McGraw-Hill.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature, 323*(6088), 533–536. https://doi.org/10.1038/323533a0.

Sakarovitch, J. (2009). Rational and recognisable power series. In M. Droste, W. Kuich, & H. Vogler (Eds.), *Handbook of Weighted Automata* (pp. 105–174). Berlin: Springer.

Salehinejad, H., Baarbe, J., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2018). Recent advances in recurrent neural networks. *CoRR*, arXiv:abs/1801.01078.

Schmitz, G. P. J., Aldrich, C., & Gouws, F. S. (1999). Ann-dt: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks, 10*(6), 1392–1401. https://doi.org/10.1109/72.809084.

Schuster, M., Paliwal, K. K., & General, A. (1997). Bidirectional recurrent neural networks. In: *IEEE Transactions on Signal Processing*.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Tech. J., 27*(4), 623–656.

Shibata, C.,& Heinz, J. (2017). Predicting sequential data with LSTMs augmented with strictly 2-piecewise input vectors. In *Proceedings of the international conference on grammatical inference*, Vol. 57 of *PMLR*, pp. 137–142.

Siegelmann, H. T., & Sontag, E. D. (1992). On the computational power of neural nets. In *Conference on computational learning theory,* pp. 440–449.

Siegelmann, H. T., & Sontag, E. D. (1994). Analog computation via neural networks. *Theoretical Computer Science, 131,* 331–360.

Steinhaus, H. (1957). Sur la division des corps matériels en parties. *Bulletin of the Polish Academy of Sciences Cl III, 4,* 801–804.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, arXiv:abs/1409.3215.

Tomita, M. (1982). Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the fourth annual conference of the cognitive science society,* pp. 105–108.

Tzeng, W.-G. (1992). A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput., 21*(2), 216–227.

van Merriënboer, B., Breuleux, O., Bergeron, A., & Lamblin, P. (2017). Automatic differentiation in ML: Where we are and where we should be going. *CORR*. https://arxiv.org/abs/1810.11530.

Verwer, S., Eyraud, R., & de la Higuera, C. (2014). PAutomaC: a probabilistic automata and hidden markov models learning competition. *Machine Learning, 96*(1–2), 129–154.

Wallace, E., Feng, S., & Boyd-Graber, J. L. (2018). Interpreting neural networks with nearest neighbors. *CoRR*, arXiv:abs/1809.02847.

Wang, Q., Zhang, K., Ororbia, A., Xing, X., Liu, X.,& Lee Giles, C. (2017). An empirical evaluation of recurrent neural network rule extraction. *Neural Computation*, *30*.

Wang, Q., Zhang, K., Liu, X., Giles, C L. (2019a). Verification of recurrent neural networks through rule extraction. In *AAAI spring symposium on verification of neural networks*.

Wang, Q., Zhang, Kaixuan., Liu, Xue., & Giles, C. Lee. (2019b). Connecting first and second order recurrent networks with deterministic finite automata. arXiv arXiv:abs/1911.04644.

Watrous, R. L., & Kuhn, G. M. (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation, 4*(3), 406–414.

Weiss, G., Goldberg, Y.,& Yahav, E. (2018a). Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the international conference on machine learning*, Vol. 80, pp. 5244–5253. PMLR.

Weiss, G., Goldberg, Y.,& Yahav, E. (2018b). On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the annual meeting of the association for computational linguistics*, pp. 740–745.

Weiss, G., Goldberg, Y.,& Yahav, E. (2019). Learning deterministic weighted automata with queries and counterexamples. In *Proceedings of NeurIPS*.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks, 1*(4), 339–356.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R.,& Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the international conference on machine learning*, vol. 37 of *PMLR*, pp. 2048–2057.