Distance and Equivalence between Finite State Machines and Recurrent Neural Networks: Computational results

Reda Marzouk[1] and Colin de la Higuera[2]

[1,2]Université de Nantes / Laboratoire des Sciences du Numérique de Nantes (LS2N UMR CNRS 6004), Nantes, France

June 3, 2020

Abstract

The need of interpreting Deep Learning (DL) models has led, during the past years, to a proliferation of works concerned by this issue. Among strategies which aim at shedding some light on how information is represented internally in DL models, one consists in extracting symbolic rule-based machines from connectionist models that are supposed to approximate well their behavior. In order to better understand how reasonable these approximation strategies are, we need to know the computational complexity of measuring the quality of approximation. In this article, we address this issue from a computational viewpoint for the case of Finite State Machines (FSMs) with Recurrent Language Models trained as Language models(RNN-LM). More precisely, we'll show the following: (a) For general RNN-LMs with a single hidden layer and a ReLu activation function: - The equivalence problem of a probabilistic deterministic finite automata (PDFA), probabilistic finite automata(PFA) and weighted finite automata (WFA) with a first-order RNN-LM is undecidable; - As a corollary, the distance problem between languages generated by PDFA/PFA/WFA and that of a RNN-LM is not recursive; -The intersection between a DFA and the cut language of an RNN-LM is undecidable; - The equivalence of a PDFA/PFA/WFA and RNN-LM in a finite support is EXP-Hard; (b) For consistent weight RNN-LMs with any computable activation function: - The Tchebychev distance approximation with PFAs -equivalently, Hidden Markov Models- is decidable; - The finite support version of the problem is at least NP-Hard. Our reduction technique from 3-SAT makes this latter fact easily generalizable to other RNN architectures (e.g. LSTMs/GRUs), and even RNNs with finite precision.

Mots-clef: Recurrent Neural Networks, Finite State Machines, Distances, Equivalence.

1 Introduction

Recurrent Neural Networks and their different variants represent an important family of Deep Learning models suitable to learning tasks with sequential data. However, just like all Deep Learning models in general, this class of models lacks interpretability, restricting its deployment in systems where security and transparency matters are of utmost importance. To tackle this issue, two major paradigms are generally explored in the literature: The first consists at raising the problem of interpretability early on the design phase of DL architectures [36][25], at the price of losing flexibility of these newly designed *transparent* architectures. A second family of techniques raises the issue of interpretability in an *ad-hoc* manner, by designing algorithmic and visualisation tools [15] to conduct interpretative analysis of already trained DL models.

Among possible *ad-hoc* strategies, one consists at extracting transparent rule-based machines from a DL trained model. the challenge is then how to convert the continuous representation of information as encoded in DL models into symbolic representations, while maintaining a good quality of prediction of these last structures.

In this work, we aim at highlighting the problem of how well finite-state models can approximate a RNN trained as language models [1] from a computational viewpoint. This problem in turn requires solving essential computational problems: can we compute

[1]We note that in language learning tasks, RNN can be trained in two ways: RNN as acceptors/classifiers [33] or RNN as language models [22]. In this article, our main focus will be on the second case.

distances between these language models? Can we decide the equivalence? These questions have received answers for PFAs [8,10,19]. We aim in this work to extend these results by including RNN language models into the picture. The class of RNNs considered in this paper are first-order RNNs with a ReLu activation function. The case of general activation functions will be discussed in the last section of the article where we give a lower bound complexity class of the approximate equivalence problem between RNN-LMs with any activation function and PFAs.

Our main results are stated as follows: (a) For general first-order RNN-LMs with ReLu activation function: 1. The equivalence problem of a PDFA/PFA/WFA and a first-order RNN-LM is undecidable; 2- As a corollary, any distance metric between languages modeled by PDFA/PFA/WFA and that of a RNN-LM is not recursive; -The intersection between a DFA and the cut-point language[27] of a RNN-LM is undecidable; - The equivalence of a PDFA/PFA/WFA and RNN-LM in a finite support is EXP-Hard; (b) For consistent first-order RNN-LMs with any computable activation function: - The Tchebychev distance approximation is decidable; - The Tchebychev distance approximation in a finite support is NP-Hard.

The rest of this article is organised as follows. Section 2 gives a concise literature overview of issues related to our problematic. Section 3 presents our results for the case of general first-order RNN language models(RNN-LM) with ReLu activation function. Section 4 is dedicated to the case of consistent RNN-LMs.

2 Related works

The problem of symbolic knowledge extraction from connectionist models is not a new issue, and one can trace back works interested in this problem since the development of the first neural architectures [23][20]. However, with the development of novel spatio-temporal connectionist models in the nineties, the most important of which is Ellman RNNs[11], and their great empirical success on inferring language models with limited amount of data and with performance results that often outscore rule-based algorithms traditionally used in the Grammatical Inference field [9], research interests in this issue has regained more attention. In fact, these works were mostly driven by a legitimate motivation: if an RNN-like structure is trained to recognise a language belonging to a given class of languages $\mathcal{C}$, and this latter can be recognised by a class of *computing devices* $\mathcal{M}$, then there must be a close connection between the representation of the target language as encoded in the RNN-like structure on one hand, and that of the corresponding computing device in $\mathcal{M}$ that is capable of recognising it on the other.

This aforementioned motivation raises two fundamental questions, at least from a theoretical viewpoint:

1. What is the expressive power of different classes of *"RNN Machines"*, as compared to classical symbolic machines (e.g. deterministic/non deterministic finite state automata, deterministic/non determinstic pushdown automata, Turing machines etc.)?
2. How can we design algorithms that extract symbolic machines from RNN models? What are the theoretical guarantees of such methods? What is the computational complexity of such problems?

We should note that these two questions are, in some sense, interrelated. If a class of RNNs is very powerful –say Turing Equivalent– computational problems related to the extraction of finite state machines are more likely to be undecidable. In fact, as a corollary of Rice's Theorem[2], the equivalence between a Turing machine and any non-trivial class of computing devices is necessarily undecidable, which means in practice that no algorithm can exist that can answer the question of equivalence between symbolic machines and "Turing- Equivalent" RNN ones. In other words, from the perspective of the theory of computation, the trade-off between expressiveness and interpretability[3] in connectionist models is unavoidable. As a consequence of the above discussion, we argue that analyzing a class of RNNs as a computational model can give many insights with regard to its interpretability.

Guided by questions raised above, we divide the rest of this section into two parts: In the first part, we examine works present in the literature that focused on the computational power of recurrent neural networks and its consequences on some computational problems concerning RNNs. In the second part, we give a brief overview of existing methods in the literature aimed at extracting finite state machines from trained RNN ones.

[2]The Rice Theorem states that any class of non-trivial languages recognised by a Turing machine is not recursive

[3]In our context, we quantify the interpretability of a model as a measure of the computational difficulty by which one can extract a finite state machine. A more rigorous formal definition of what is an *intepretable model* is still an arguably open question.

2.1 Computational power of RNNs

The question of the computational capabilities of different classes of RNN has been addressed since the early development of neural systems. To the best of the author's knowledge, early works that addressed this dates back to the middle of the previous century by Kleene [17], where it was proven that networks with binary threshold activation functions are capable of implementing finite state automata. In [26], Pollack designed a *Turing-complete* class of high-order recurrent neural networks with two types of activation function (linear and Heaviside). This result was later extended in [29], where authors relaxed the high-order requirement, and showed that first-order RNNs with saturated-linear activation functions were *Turing Complete*. Later on, Kilian *et al.* generalised this result to sigmoidal activation functions [16].

The Turing Completeness of some classes of RNNs has many consequences with respect to the computational class to which belong many problems related to them. In [7], the authors proved that the problem of deciding whether a RNN language model -RNN-LM- with ReLu activation function is consistent (encodes a valid probability distribution) or not is an undecidable problem. Moreover, the consensus string problem and finding a minimal RNN-LM equivalent to a given RNN-LM or testing the equivalence between two RNN-LMs are also undecidable.

Given these pessimistic results about computability of several important problems related to RNNs, a new line of research suggests to analyze the practical capabilities computational power of neural nets instead of the classical *"unrealistic"* theoretical model, by constraining the amount of memory resources of the RNN hidden units to be finite [34][21]. Under this constraint, Korskky *et al.* [18] proved that RNNs with one hidden layer and ReLu activation, and GRUs are expressively equivalent to deterministic finite automata. In [34], Weiss *et al.* showed that the class of finite precision LSTMs were able to simulate counter machines, while the simple class of Elman RNNs and GRUs can't.

2.2 Extraction of automata-based machines from trained RNNs

Early works investigating the problem of extracting automata-based machines from trained RNNs coincide with the emergence of novel RNN architectures [11][12] in early nineteens that have shown promising results for the task of inferring language models from limited data. These early works have mainly focused on the extraction of deterministic finite automata (DFAs) from RNNs trained to recognise regular languages, and most of which were based on the assumption that a well-trained RNN to recognise a regular language tend to group hidden states of the RNN into clusters that maps directly to states of the minimal DFA recognising the target regular language. Based on this assumption, the problem of DFA extraction from RNNs boils down to a clustering/quantization problem of the RNN's hidden state space, and many clustering techniques were proposed for this task: Quantization by Equipartition [13][32], Hierarchical Clustering [1], k-means [35][28], fuzzy clustering [6] etc.

During the last few years, as RNN-based architectures became more sophisticated and thus harder to be a subject of interpretative analysis, the issue has gained an increasing interest among researchers, and new methods were proposed in the literature to extract automata-based machines from modern classes of RNNs, such as LSTM/GRUs. In [33], Weiss *et al.* proposed an adaptation of the $L*$ algorithm [2] to extract deterministic finite automata (DFA) from RNN Acceptors (LSTMs/GRUs), where an RNN Acceptor model serves as a black box oracle for *approximate* equivalence and membership queries, hinting that the exact equivalence query is *"likely to be intractable"*. The same authors extended their work in [33] to extract Probabilistic Deterministic Finite Automata from RNN-LMs. In order to answer equivalence queries, authors used a sampling strategy of both models, and gave theoretical guarantees of its convergence in probability under a relaxed notion of equivalence. Ayache *et al.* [3] employed the spectral learning framework [4] to extract Weighted Finite Automata (WFA) from a RNN language model. In [24], Okudono *et al.* raised the problem of answering the equivalence query between a RNN language model and a WFA proposing an empirical regression-based technique to perform this task. However, no theoretical guarantees were provided to back their method.

Many methods mentioned theoreof rely on the equivalence query oracle as an internal component in the algorithmic design. Our work aims at shedding light on the computational difficulty of designing such oracle. The problem of deciding the equivalence between finite state machines and non-trivial class of RNNs is believed to be intractable [34]. In this work, we'll give a formal proof that the equivalence problem is indeed undecidable even in the case of simple first-order RNNs with ReLu activation function with different weighted finite state machines. Facing this negative result, we address the question of whether approximate equivalence in the sense of Tchebychev distance is tractable when the trained RNN-LM is guaranteed to be consistent. We prove that the approximate equivalence problem is decidable, and show that the lower bound complexity class on this problem with general activation functions is NP-Hard. We argued that this lower bound complexity result holds also for other architectures of interest, such as LSTM/GRUs.

3 Definitions and Notations

Let Σ be a finite alphabet. The set of all finite strings is denoted by Σ^*. The set of all strings whose size is equal (resp. greater than or equal) to n is denoted by Σ^n(resp. $\Sigma^{\geq n}$). For any string $w \in \Sigma^*$, the size of w is denoted by $|w|$, and its n-th symbol by w_n. The prefix of length n for any string $w \in \Sigma^{\geq n}$ will be referred to as $w_{:n}$. The symbol \$ denotes a special marker. The symbol $\Sigma_\$$ will refer to the set $\Sigma \bigcup \{\$\}$.

Weighted languages: A weighted language f over Σ is a mapping that assigns to each word $w \in \Sigma^*$ a weight $f(w) \in \mathbb{R}$. A WL f is called consistent, if it encodes a valid probability distribution, i.e. satisfies the following properties: $\forall w \in \Sigma^* : f(w) \geq 0$, $\sum\limits_{w \in \Sigma^*} f(w) = 1$. Two WLs f_1, f_2 are said to be equivalent if: $\forall w \in \Sigma^* : \ f_1(w) = f_2(w)$. The Tchebychev distance metric between two WLs is denoted $d_\infty(f_1, f_2)$, and defined as: $\max\limits_{w \in \Sigma^*} |f_1(w) - f_2(w)|$. Finally, we define, for a given scalar $c > 0$, the cut-point language of f with respect to c and denoted $\mathcal{L}_{f,c}$, as the set of finite words whose values are greater or equal to c.

In Section 4, a 3-SAT formula will be denoted by the symbol Ψ. A formula is comprised of n Boolean variables denoted $x_1, ..x_n$, and k clauses $C_1, ..C_k$. For each clause, we'll use notation $l_{i1},\ l_{i2},\ l_{i3}$ to refer to its three composing literals. For a given string $w \in \{0,1\}^n$, the number of clauses satisfied by w will be denoted by N_w.

For the rest of this section, we shall provide a formal definition of the class of first-order RNN-LMs that we'll study in this work, and basic definitions of different automata-based machines that we'll encounter throughout the rest of this article.

Definition 3.1. *[7] A First-order RNN Language model is a weighted language $R : \Sigma^* \to \mathbb{R}$ and is defined by the tuple $< \Sigma, N, h^{(0)}, \sigma, W, (W')_{\Sigma_\$}, E, E' >$ such that:*

- *Σ is the input alphabet,*
- *N the number of hidden neurons,*
- *$h^{(0)} \in \mathbb{Q}^N$ is the initial state vector,*
- *$f : \mathbb{Q} \to \mathbb{Q}$ is a computable activation function,*
- *$W \in \mathbb{Q}^{N \times N}$ is the transition matrix,*
- *$\{W'_\sigma\}_{\sigma \in \Sigma_\$}$, where each $W'_\sigma \in \mathbb{Q}^N$ is the embedding vector of the symbol $\sigma \in \Sigma_\$$,*
- *$O \in \mathbb{Q}^{\Sigma_\$ \times N}$ is the output matrix,*
- *$O' \in \mathbb{Q}^{\Sigma_\$}$ the output bias vector.*

The computation of the weight of a given string w by R is given as follows. (a) Recurrence equations:

$$h^{(t+1)} = f(W.h^{(t)} + W'_{w_t})$$

$$E_{t+1} = Oh^{(t+1)} + O'$$

$$E'_{t+1} = softmax_2(E_{t+1})$$

(b) The resulting weight:

$$R(w) = \prod_{i=0}^{|w|+1} E'_i$$

where $w_0 = w_{|w|+1} = \$$

Remark that, in order to avoid uncomputability issues, we used softmax base 2 defined as: $softmax_2(x)_i = \frac{2^{x_i}}{\sum\limits_{j=1}^{n} 2^{x_j}}$ for any $x \in \mathbb{R}^d$ instead of the standard softmax in the previous definition. In the following, hidden units of the network will be designated by lowercase letters $n_1, n_2, ..$, and their activations at time t by h_n^t. Also, we denote by $\mathcal{R}_f$ the class of RNN-LMs when f is the activation function. For example, an important class of RNN-LMs that will be used extensively in the rest of the article is $\mathcal{R}_{ReLu}$.

Weighted Finite Automata (WFA). WFAs represent weighted versions of nondeterministic finite automata, where transitions between states, denoted $\delta(q, \sigma, q')$ where $q, q' \in Q$ represents states of the WFA are labelled with a rational weight $T(q, \sigma, q')$, and each of its nodes $q \in Q$ is labelled by a pair of rational numbers $(I(q), P(q))$ that represents respectively the initial-state and final-state weight of q. WFAs model weighted languages where the weight of a string w is equal to the sum of the weights of all paths whose transitions encode the string w. The weight of a path p is calculated as the product of the weight labels of all its transitions, multiplied by the initial-state weight of its staring node and the final-state weight of its ending node.

Probabilistic Finite Automata (PFA). A PFA is a WFA with two additional constraints: First, the sum of initial-state weights of all states is a valid probability distribution over the state space. Second, for each state, the sum of weights of its outcoming edges added to its finite-state weight is equal to 1. This additional constraint restricts the power of PFAs to encode stochastic languages [30], which makes it useful for representing language models. Interestingly, PFAs are proven to be equivalent to Hidden Markov Models (HMMs), and the construction of equivalent HMMs from PFAs and vice versa can be done in polynomial time[31]. The deterministic version of PFAs, a.k.a Deterministic Probabilistic Finite Automata (DPFA), enforces the additional constraint that for any state q, and for any symbol σ there is at most one outgoing transition labelled by σ from q. We note that there is a strict inclusion in terms of expressive power between PDFA, PFA and WFAs [31].

4 Computational results for general RNN-LMs with ReLu activation functions

The choice of $ReLu$ in this part of the article is not arbitrary. In fact, due to its nice piecewise-linear property and its wide use in practice, the $ReLu(.)$ function is first choice to analyze theoretical properties of RNN architectures. Analyzing the case of RNNs with highly non-linear activation functions (e.g. the sigmoid, the hyperbolic tangent etc.), is left for future research.

4.1 Turing Completeness of RNNs with ReLu: Siegelmann's construction

The basic building block for proving computational results presented in this part of the article is the work done by Siegelmann and al. in [29] to prove the Turing completeness of a certain class of first-order RNNs. Hence, we propose, in this section, to provide a global scope of this construction, followed by an equivalent reformulation of their main theorem that will be relevant for our work.

The main intuition of Siegelmann *et al.*'s work is that, with an appropriate encoding of binary strings, a first-order RNN with a saturated linear function can readily simulate a stack data structure by making use of a single hidden unit. For this, they used 4-base encoding scheme that represents a binary string w as a rational number: $Enc(w) = \sum_{i=1}^{|w|} \frac{w_i}{4^i}$. Backed by this result, they proved than any two-stack machine can be simulated by a first-order RNN with linear saturated function, where the configuration of a running two-stack machine (i.e. the content of the stacks and the state of the control unit) is stored in the hidden units of the constructed RNN. Finally, given that any Turing Machine can be converted into an equivalent two-stack machine (The set of two-stack machines is Turing-complete [14]), they concluded their result.

In the context of our work, two additional remarks need to be noted about Siegelmann's construction: - First, although the class of first-order RNNs examined in their work uses the saturated linear function as an activation function, as raised in [7], their result is generalizable to the ReLu activation function (or, more generally, any computable function that is linear in the support [0,1])? - Second, although not mentioned in their work, the construction of the RNN from a Turing Machine is polynomial in time. In fact, on one hand, the number of hidden units of the constructed RNN is linear in the size of the Turing Machine, and the construction of transition matrices of the network is also linear in time. On the other hand, notice that the 4-base encoding map $Enc(.)$ is also computable in linear time.

In light of these remarks, we are now ready to present the following theorem:

Theorem 4.1. *(Theorem 2, [29]) Let $\phi : \{0,1\}^* \to \{0,1\}^*$ be any computable function, and M be a Turing Machine that implements it. We have, for any binary string w, there exists $N = O(poly(|M|))$, $h^{(0)} = [Enc(w) \quad 0..0] \in \mathbb{Q}^N$, $W \in \mathbb{Q}^{N\times N}$, such that for any finite alphabet Σ, $\forall \sigma \in \Sigma_\$:\ W'_\sigma \in \mathbb{Q}^N, O \in \mathbb{Q}^{|\Sigma_\$|\times N},\ O' \in \mathbb{Q}^{|\Sigma_\$|}$, $R =< \Sigma, N, ReLu, W, W', O, O' >\in \mathcal{R}_{ReLu}$ verifies:*

- *if $\phi(w)$ is defined, then there exists $T \in \mathbb{N}$ such that the first element of the hidden vector h_T is equal to $Enc(\phi(w))$, and the second element is equal to 1,*
- *if $\phi(w)$ is undefined (i.e. M never halts on w), then for all $t \in \mathbb{N}$, the second element of the hidden vector h_t is always equal to zero.*

Moreover, the construction of h_0 and W is polynomial in $|M|$ and $|w|$.

In the following, we'll denote by $\mathcal{R}^{M,w}_{ReLu}$ the set of RNNs in $\mathcal{R}_{ReLu}$ that simulate the TM M on w. It is important to note that the construction of a RNN that simulates a TM on a given string in the previous theorem is both input and output independent. The only constraints that are enforced by the construction are placed on a block of the transition matrix of the network, and the initial state. In fact, the input string is *placed* in the first stack of the two-stack machine before running the computation (i.e. in the initial state $h^{(0)}$). Under this construction, the first stack of the machine

is encoded in the first hidden unit of the network. Afterwards, the *RNN Machine* runs on the empty string, and halts (If It ever halts) when the halting state of the machine is reached. In theorem 1.1, the halting state of the machine is represented by the second neuron of the network. In the rest of the article, we'll refer to the neuron associated to the halting state by the name *halting neuron*, denoted n_{halt}.
We present the following corollary that gives a characterization of the halting machine problem[4] that relates it to the class $\mathcal{R}_{ReLu}$:

Corollary 4.2. *Let M be any Turing Machine, and w be a binary string, M halts on w if and only if for any $R \in \mathcal{R}_{Relu}^{M,w}$, there exists $T \in \mathbb{N}$, such that $\forall t < T$: $h_{n_{halt}}^{(t)} = 0$, and $h_{n_{halt}}^{(T)} = 1$.*

4.2 The equivalence problem between FSAs and general RNNs

The equivalence problem between a DPFA and a general RNN-LMs is formulated as follows:
Problem. Equivalence Problem between a DPFA and a general RNN
Given a general RNN-LM $R \in \mathcal{R}_{ReLu}$ and a DPFA $\mathcal{A}$. Are they equivalent?

Theorem 4.3. *The equivalence problem between a DPFA and a general RNN is undecidable*

Proof. We'll reduce the halting Turing Machine problem to the Equivalence problem. Let $\Sigma = \{a\}$. We first define the trivial DPFA $\mathcal{A}$ with one single state q_0, and $T(\delta_{q_0,a,q_0}) = P(q_0) = \frac{1}{2}$, $I(q_0) = 1$. This DPFA implements the weighted language $f(a^n) = \frac{1}{2^{n+1}}$.
Let M be a Turing Machine and $w \in \Sigma^*$. We construct $R \in \mathcal{R}_{ReLu}^{M,w}$ such that $O[n_{halt}, a] = 1$,0 everywhere and O' is equal to zero everywhere. We build another RNN R' from R by adding one neuron in its hidden layer, denoted n' such that: $h_{n'}^{(0)} = 0$, $\forall t \geq 0$: $h_{n'}^{(t+1)} = ReLu(h_{n'}^{(t)})$, $O[n', \$] = 1$.
Notice that, by Corollary 4.2, the TM M never halts on w if and only if $\forall T$: $(h_{n_{halt}}^{(T)}, h_{n'}^{(T)}) = (0,0)$, i.e. $R'(a^n) = \frac{1}{2^{n+1}}$. That is, the TM M doesn't halt on w if and only if the DPFA $\mathcal{A}$ is equivalent to R', which completes the proof. □

A direct consequence of the above theorem is that the equivalence problem between PFAs/WFAs and general RNN-LMs in $\mathcal{R}_{ReLu}$ is also undecidable, since the DPFA problem case is immediately reduced to the general case of PFAs (or WFAs). Another important consequence is that no distance metric can be computed between DPFA/PFA/WFA and $\mathcal{R}_{ReLu}$:

Corollary 4.4. *Let $\Sigma = \{a\}$. For any distance metric d of Σ^*, the total function that takes as input a description of a PDFA $\mathcal{A}$ and a general RNN-LM $\mathcal{R}_{ReLu}$ and outputs $d(\mathcal{A}, R)$ is not recursive.*
This fact is also true for PFAs and WFAs.

Proof. The proof relies on the properties of distance metrics. □

4.3 Intersection of the cut language of a general RNN-LM with a DFA

In this subsection, we are interested in the following problem:
Problem. Intersection of a DFA and the cut-point language of a general RNN-LM
Given a general RNN-LM $R \in \mathcal{R}_{ReLu}$, $c \in \mathbb{Q}$, and a DFA $\mathcal{A}$, is $\mathcal{L}_{R,c} \bigcap \mathcal{L}_{\mathcal{A}} = \emptyset$? Before proving that this problem is undecidable, we shall recall first a result proved in [7]:

Theorem 4.5. *(Theorem 9, [7]) Define the highest-weighted string problem as follows: Given a RNN-LM $R \in \mathcal{R}_{ReLu}$, and $c \in (0,1)$: Does there exist a string w such that $R(w) > c$?*
The highest-weighted string problem is undecidable.

Corollary 4.6. *The intersection problem of a DFA and the cut-point language of a general RNN-LM is undecidable.*

Proof. We shall reduce the highest-weighted string problem from the intersection problem. Let $R \in \mathcal{R}_{ReLu}$ a general weighted RNN-LM, and $c \in (0,1)$. Construct the automaton $\mathcal{A}$ that recognizes Σ^*. We have that $\mathcal{L}_{\mathcal{A}} \bigcap \mathcal{L}_R = \mathcal{L}_R = \emptyset$ if and only if there exist no string w such that $R(w) > c$, which completes the proof. □

4.4 The equivalence problem over finite support

Given that the equivalence problem between a general RNN-LM and different classes of finite state automata is undecidable, a less ambitious goal is to decide whether a RNN-LM agrees with a finite state automaton over a finite support. We formalize this problem as follows:
Problem. The EQ-Finite problem between PDFA and

[4]The Halting Machine problem is defined as follows: Given a TM M and a string w, does M halt on w? This problem is undecidable.

general RNN-LMs
Given a general RNN-LM $R \in \mathcal{R}_{ReLu}$, $m \in \mathbb{N}$ and a PDFA $\mathcal{A}$. Is R equivalent to $\mathcal{A}$ over $\Sigma^{\leq m}$?

Theorem 4.7. *The EQ-Finite problem is EXP-Hard.*

Proof. We reduce the bounded halting problem [5] to the EQ-Finite problem.
The proof is similar to the used for Theorem 4.3. We are given a TM M, a string w and $m \in \mathbb{N}$. Let $\Sigma = \{a\}$. We construct a general RNN-LM R' by augmenting $R \in \mathcal{R}^{M,w}_{ReLu}$ with a neuron n' as in Theorem 4.3. By Theorem 4.1, this reduction runs in polynomial time. On the other hand, let $\mathcal{A}$ be the trivial PDFA with one single state q_0, and $T(\delta_{q_0,a,q_0}) = P(q_0) = \frac{1}{2}$, $I(q_0) = 1$. Note that R' doesn't halt in m steps if and only if $\forall T \leq m$: $(n^{(T)}_{halt}, n'^{(T)}) = (0,0)$, i.e. $R'(a^n) = \frac{1}{2^{n+1}}$ for the first m running steps on R', in which case the language modelled by R' is equal to f in $\Sigma^{\leq m}$. Hence, $\mathcal{A}$ is equivalent to R in $\Sigma^{\leq m}$ if and only if M doesn't halt on the string w in less or equal than m steps. □

5 Computational results for consistent RNN-LMs with general activation functions

In the previous section, we have seen that problems related to equivalence and distance in the general case turned out to be either undecidable, or intractable when restricted to finite support. In this section, we examine the case where trained RNN-LMs are guaranteed to be consistent[6], and we raise the question of approximate equivalence between PFAs and first-order consistent RNN-LMs with *general* computable activation functions. It's worth noting that all results holding here for PFAs remain valid for Hidden Markov Models(HMM), since HMMs and PFAs are proved to be equivalent and there exists polynomial time algorithms to convert one to another and vice versa (See Propositon 4-5, [31]). For any computable activation function σ, we formalise this question in the following two decision problems:

Problem. *Approximating the Tchebychev distance between RNN-LM and PFA*
Instance: A consistent RNN-LM $R \in \mathcal{R}_\sigma$, a PFA $\mathcal{A}$, $c > 0$
Question: Does there exist $|w| \in \Sigma^*$ such that $|R(w) - \mathcal{A}(w)| > c$

Problem. *Approximating the Tchebychev distance between consistent RNN-LM and PFA over finite support*
Instance: A consistent RNN $R \in \mathcal{R}_\sigma$, a PFA $\mathcal{A}$, $c > 0$ and $N \in \mathbb{N}_+$,
Question: Does there exist $|w| \leq N$ such that $|R(w) - \mathcal{A}(w)| > c$
Note that there is no constraint on the activation function used for consistent RNN-LMs in these defined problems, provided it is computable. The first fact is easy to prove:

Theorem 5.1. *Approximating the Tcheybechev distance between RNN-LM and PFA is decidable.*

Proof. Let R be a consistent RNN-LM and $\mathcal{A}$ be a PFA. An algorithm that can decide this problem runs as follows: enumerate all strings $w_1, ..w_t, ..$ in Σ^* until we reach a string that satisfies this property in which case the algorithm returns Yes. If there is no such string, by definition of consistency, there will be a finite time T[7] such that $\sum_{t=1}^{T} R(w_t) \geq 1 - c$, $\sum_{t=1}^{T} \mathcal{A}(w_t) \geq 1 - c$ in which case, we have: $\forall t > T$: $R(w_t) < c$ and $\mathcal{A}(w_t) < c$ which implies $\forall t > T : |R(w_t - \mathcal{A}(w_t)| < c$. When T is reached, the algorithm returns No. □

5.1 Approximating the Tcheybetchev distance over a finite support

The rest of this section is dedicated to the proof of NP-Hardness result. And, we'll derive below the construction of a PFA and a RNN from a given 3-SAT formula which will help us prove the result. Without loss of generality, we assume for the rest that literals $\{l_{i1}, l_{i2}, l_{i3}\}$ of a given clause C_i are arranged in the order of atoms from which they derive, and we denote by $i^*_{C_i}$ the index of the atom of l_{i1}. Let $\epsilon \in (0, \frac{1}{2})$ whose value will be specified later.

- **Construction of a PFA $\mathcal{A}$:** The construction of our PFA is inspired from the work done in [5], and illustrated in Figure 1. Intuitively, each clause i in Ψ is represented by two paths in the PFA, one that encodes a satisfiable assignment of the variables for this clause,

[5] The bounded halting problem is defined as follows: Given a TM M, a string x and an integer m, encoded in binary form. Decide if M halts on x in at most n steps? This problem is EXP-Complete.

[6] It was proven until recently that RNN-LMs with ReLu activation function are not necessary consistent, and deciding consistency is undecidable[7]. Characterizing consistency of different classes of RNN-LMs is still an open problem.

[7] T can be determined while running the algorithm through summing the probabilities of all reached strings in the enumeration

and the other not. More formally, the PFA $\mathcal{A}$ is defined as:

- $Q_{\mathcal{A}} = \{q_0\} \cup \{q_{ij}^c : \ i \in [1,k], \ j \in [1,n], \ c \in \{T,F\}\}$ is the set of states,
- Initial-state weights: $I_{\mathcal{A}}(q_0) = 1, \ 0$ otherwise,
- Final-state weights:
  - For each clause i: $P_{\mathcal{A}}(q_{in}^N) = 1 - 2\epsilon$
  - All the other states in $\mathcal{A}$ has a final-state probability equal to 2ϵ
- Transitions: For each clause C_i,
- $\forall S \in \{T,F\}, \ a \in \Sigma$: $(q_0, a, q_{i,1}^S) = \frac{1}{2k} - \frac{\epsilon}{k}$
- If $i_{C_i}^* \neq 1$:
  - If $l_{i1} = x_{i_{C_i}^*}$:
    $\forall S \in \{T,F\} : \ T_{\mathcal{A}}(q_{i_{C_i}^*-1}^S, 1, q_{i_{C_i}^*}^T) = \frac{1}{2} - \epsilon$ and $T_{\mathcal{A}}(q_{i_{C_i}^*-1}^S, 0, q_{i_{C_i}^*}^F) = \frac{1}{2} - \epsilon$
  - else: $\forall S \in \{T,F\} : \ T_{\mathcal{A}}(q_{i_{C_i}^*-1}^S, 0, q_{i_{C_i}^*}^T) = \frac{1}{2} - \epsilon$ and $T_{\mathcal{A}}(q_{i_{C_i}^*-1}^S, 1, q_{i_{C_i}^*}^F) = \frac{1}{2} - \epsilon$
  - If $i_{C_i}^* > 2$, then:
    $\forall 1 \leq i < i_{C_i}^* - 1, \ a \in \Sigma, \ S \in \{T,F\}$:, we have: $T_{\mathcal{A}}(q_i^S, a, q_{i+1}^S) = \frac{1}{2} - \epsilon$
- Else:
  - If $l_{i1} = x_1$: $\forall a \in \Sigma : \ T_{\mathcal{A}}(q_{i,1}^T, a, q_{i,2}^T) = \frac{1}{2} - \epsilon$. And:
    * If $x_2 = l_{i2}$: then $T_{\mathcal{A}}(q_{i1}^F, 1, q_{i2}^T) = \frac{1}{2} - \epsilon$, and $T_{\mathcal{A}}(q_{i1}^F, 0, q_{i2}^F) = \frac{1}{2} - \epsilon$,
    * If $l_{i2} = \bar{x}_2$: then $T_{\mathcal{A}}(q_{i1}^F, 0, q_{i2}^T) = \frac{1}{2} - \epsilon$, and $T_{\mathcal{A}}(q_{i1}^N, 1, q_{i2}^N) = \frac{1}{2} - \epsilon$
    * Otherwise $\forall a \in \Sigma : T_{\mathcal{A}}(q_{i1}^F, a, q_{i2}^F) = \frac{1}{2} - \epsilon$
  - else: $\forall a \in \Sigma : \ (q_{i1}^F, a, q_{i2}^T) \in \delta_{\mathcal{A}}$, and:
    * If $x_2 \in \{l_{i1}, \ l_{i2}, \ l_{i3}\}$, then $T_{\mathcal{A}}(q_{i1}^T, 1, q_{i2}^T) = \frac{1}{2} - \epsilon$, and $(q_{i1}^T, 0, q_{i2}^F) = \frac{1}{2} - \epsilon$,
    * If $\bar{x}_2 \in \{l_{i1}, \ l_{i2}, \ l_{i3}\}$, then $T_{\mathcal{A}}(q_{i1}^T, 0, q_{i2}^T) = \frac{1}{2} - \epsilon$, and $T_{\mathcal{A}}(q_{i1}^T, 1, q_{i2}^F) = \frac{1}{2} - \epsilon$
    * Otherwise, $\forall a \in \Sigma : \ T_{\mathcal{A}}(q_{i1}^T, a, q_{i2}^F) = \frac{1}{2} - \epsilon$
- For $i_{C_i}^* \leq i < n$:
  - $\forall a \in \Sigma : \ T_{\mathcal{A}}(q_i^T, a, q_{i+1}^T) = \frac{1}{2} - \epsilon$,
  - If $x_i \in \{l_{i1}, l_{i2}, l_{i3}\}$:
    * $T_{\mathcal{A}}(q_i^F, 1, q_{i+1}^T) = \frac{1}{2} - \epsilon$,
    * $T_{\mathcal{A}}(q_i^F, 0, q_{i+1}^F) = \frac{1}{2} - \epsilon$,
  - Else if $\bar{x}_i \in \{l_{i1}, l_{i2}, l_{i3}\}$:
    * $T_{\mathcal{A}}(q_i^F, 0, q_{i+1}^T) = \frac{1}{2} - \epsilon$,
    * $T_{\mathcal{A}}(q_i^F, 1, q_{i+1}^F) = \frac{1}{2} - \epsilon$,
    - Else: $\forall S \in \{T,F\}, \ a \in \Sigma : \ T_{\mathcal{A}}(q_i^S, a, q_{i+1}^S) = \frac{1}{2} - \epsilon$

The construction above runs in $O(nk)$ time.

• **Construction of a RNN:** The RNN R we'll construct is trivial, and it generates the quantitative language $R(w) = 2(\frac{1}{2} - \epsilon)^{|w|}\epsilon$. More formally, our RNN is defined as:

- N = 2 (2 hidden neurons),
- $\begin{pmatrix} h_{n_1}^{(0)} \\ h_{n_2}^{(0)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
- Transition matrices: $W_{in} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$; $W_0 = W_1 = W_\$ = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
- Output matrices: $O = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$, $O' = \begin{pmatrix} \log_2 \frac{1-2\epsilon}{4\epsilon} \\ \log_2 \frac{1-2\epsilon}{4\epsilon} \\ 0 \end{pmatrix}$

  where $\log_2(.)$ is the logarithm to the base 2

What's left is to show that $R(w) = 2(\frac{1}{2}$ - $\epsilon)^{|w|}$ defines a consistent language model:

Proposition 5.2. *For any $\epsilon < \frac{1}{2}$, the weighted language model defined as $f(w) = 2(\frac{1}{2} - \epsilon)^{|w|}\epsilon$ is consistent.*

Proof. We have:

$$\sum_{w \in \Sigma^*} f(w) = 2\epsilon \sum_{n \in \mathbb{N}} \sum_{w:|w|=n} (\frac{1}{2} - \epsilon)^n$$
$$= 2\epsilon \sum_{n \in \mathbb{N}} (1 - 2\epsilon)^n$$

By applying the equality: $\sum_{n \in \mathbb{N}} x^n = \frac{1}{1-x}$ for any $|x| < 1$ on the sum present in the right-hand term of the equation above, we obtain the result. ☐

Proposition 5.3. *Let Ψ be an arbitrary 3-SAT formula with n variables and k clauses. Let $\mathcal{A}$ be the PFA constructed from Ψ by the procedure detailed above, the probabilistic language generated by $\mathcal{A}$ is given as:*

$$\mathcal{A}(w) = \begin{cases} 2(\frac{1}{2} - \epsilon)^{|w|}\epsilon & if \ |w| < n \\ 2(\frac{1}{2} - \epsilon)^{|w|}\epsilon[\frac{N_w}{k}\frac{1-2\epsilon}{2\epsilon} + \frac{k-N_w}{k}] & if \ |w| = n \\ 2(\frac{1}{2} - \epsilon)^{|w|}\epsilon[\frac{N_{w:n}}{k}\frac{2\epsilon}{1-2\epsilon} + \frac{k-N_{w:n}}{k}] & else \end{cases}$$

Proposition 5.4. *For any rational number $\epsilon < \frac{1}{4}$, there exists a rational number c_ϵ such that Ψ is satisfiable if and only if $d_\infty(R, \mathcal{A}) > c_\epsilon$*

Proof. For any w such that $|w| < n$, $|R(w) - \mathcal{A}(w)| = 0$.

For $|w| = n$, we have:

$$|R(w) - \mathcal{A}(w)| = 2\epsilon(\frac{1}{2} - \epsilon)^n \frac{N_w}{k}(\frac{1-4\epsilon}{2\epsilon})$$

On the other hand, for $|w| > n$, we have:

$$|R(w) - \mathcal{A}(w)| = 2\epsilon(\frac{1}{2} - \epsilon)^{|w|} \frac{N_w}{k}\frac{1-4\epsilon}{1-2\epsilon}$$

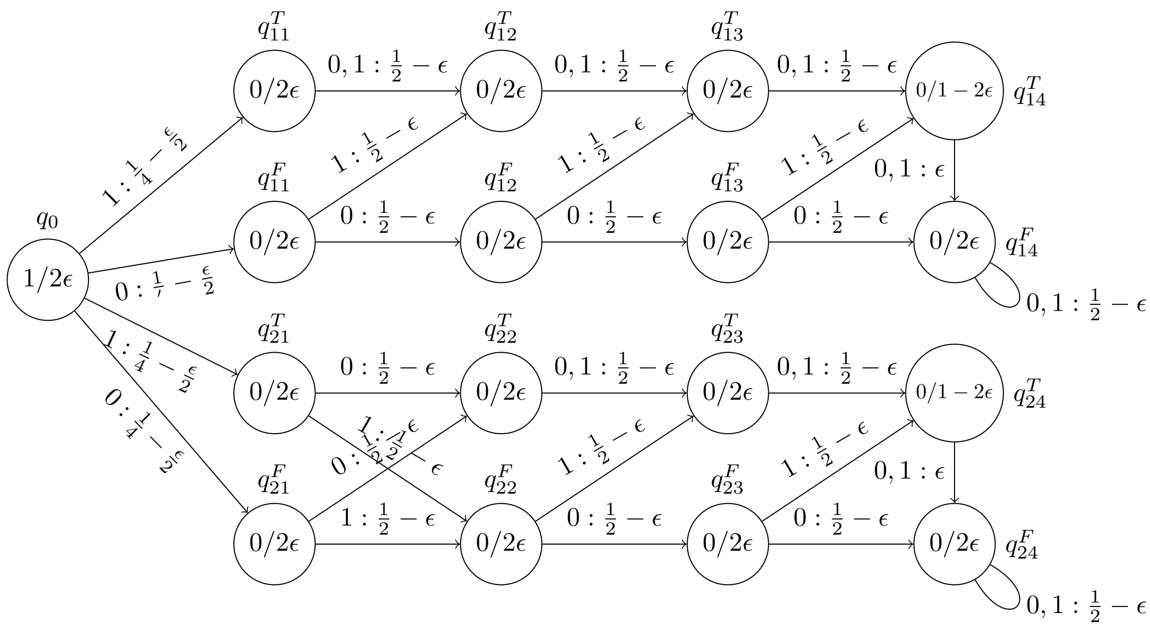

Figure 1: A graphical representation of the PFA constructed from $\Psi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$

Note that we have for any $\epsilon < \frac{1}{4}$:

$$\forall w \in \Sigma^{\geq n} : \quad |R(w) - \mathcal{A}(w)| \leq |R(w_{:n}) - \mathcal{A}(w_{:n})|$$

This means that, under this construction, the maximum is reached necessarily by a string whose length is exactly equal to n. Thus, we obtain:

$$d_\infty(R, \mathcal{A}) = 2\frac{\epsilon}{k}(\frac{1}{2} - \epsilon)^n \frac{1 - 4\epsilon}{2\epsilon} \max_{w \in \Sigma^n} N_w$$

Note that Ψ is satisfiable if and only if $\max_{w \in \Sigma^n} N_w = k$. As a result, pick any $s \in [k-1, k)$, and define $c_\epsilon = 2\frac{\epsilon s}{k}(\frac{1}{2} - \epsilon)^n \frac{1-4\epsilon}{2\epsilon}$, the formula is satisfiable if and only if $d_\infty(R, \mathcal{A}) > c_{epsilon}$. □

Theorem 5.5. *The Tchebychev distance approximation problem between consistent RNN-LMs and PFAs in finite support is NP-Hard.*

Proof. We reduce the 3-SAT satisfiability problem to our problem. Let Ψ be an arbitrary 3-SAT formula. Construct a PFA $\mathcal{A}$ and a RNN R as specified previously. Choose a rational number $\epsilon < \frac{1}{4}$. Let $c_\epsilon > 0$ be any rational number as specified in the proof of Proposition 5.4, and $N = n + 1$. By Proposition 5.4, Ψ is satisfiable if and only if $d_\infty(R, \mathcal{A}) > c_\epsilon$, which completes the proof □

Remarks:

NP-Hardness for LSTMs/GRUS: Although our main focus in this article was on first-order RNN-LMs with one hidden layer, It is worth noting that the NP-Hardness reduction technique from 3-SAT problem we employed can easily be generalized to the case of LSTMs, and GRUs, two widely used RNN architectures in practice. Indeed, our reduction relies on the construction of a *memoryless* first-order RNN which makes abstraction of the state of the hidden units of the network, and exploits only the output bias vector O'. Hence, provided we have first-order output function for a LSTM (or GRU) architecture, the NP-Hardness result demonstrated above is easy to extend our proof to these architectures.

Finite precision RNNs: As said earlier in section 2, a new line of work considered the analysis of the computational power of RNNs with bounded resources, which is a realistic condition in practice[21][34]. Broadly speaking, a finite-precision RNN is one whose weights and values of its hidden units are stored using a finite number of bits (See [18] for further details). Under our construction, the same remark raised above about LSTMs/GRUs can be applied to RNNs with finite precision. In fact, It's easy to notice that, with a judicious choice of ϵ, say $\frac{1}{10}$ (in which case $O'[0] = O'[1] = 2$), the *toy* memoryless RNN we constructed in the proof requires only 2 bits to encode a hidden unit and a weight value of the network. This shows that even approximating the Tcheybetchev distance in finite support between a language represented by PFA and that of a finite-precision first-order RNN with any computable activation function is also NP-Hard.

6 Conclusion and perspectives

In this article, we investigated some computational problems related to the issue of approximating trained RNN language models by different classes of finite state automata. We proved that the equivalence problem of PDFAs/PFAs/WFAs and general weighted first-order RNN-LM with ReLu activation function with a single hidden layer is generally undecidable, and, as a result, trying to calculate any distance between them can't be computed. When restricting RNN-LMs to be consistent, we proved that approximating the Tchebychev distance between consistent RNN-LMs with general computable activation functions and PFAs/HMMs is decidable, and that the same problem when restricted to a finite support is at least NP-Hard. Moreover, we gave arguments that the reduction strategy from 3-SAT problem we employed to prove this latter result makes this result generalizable to the class of LSTMs/GRUs and finite precision RNNs.

This work provides first theoretical results of examining equivalence and the quality of approximation problems between automata-based models and RNNs from a computational viewpoint. Yet, there are still many interesting problems on the issue that could motivate future research, such as: Is the equivalence problem between general RNN-LMs and different classes of finite state machines still undecidable when other highly non-linear activation functions (e.g. sigmoid, hyperbolic tangent ...) are used instead of ReLus? Is the equivalence problem between the cut-point language of an RNN-LM and a DFA decidable? If an RNN-LM is trained to recognise a language generated by a regular grammar, can we decide if its cut-point language is indeed regular? etc.

References

[1] Alquezar, R. and Sanfeliu, A. (1994). A hybrid connectionist-symbolic approach to regular grammatical inference based on neural learning and hierarchical clustering. In Carrasco, R. C. and Oncina, J., editors, *Grammatical Inference and Applications*, pages 203–211. Springer.

[2] Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106.

[3] Ayache, S., Eyraud, R., and Goudian, N. (2019). Explaining black boxes on sequential data using weighted automata. In *Proceedings of The 14th International Conference on Grammatical Inference 2018*, volume 93 of *Proceedings of Machine Learning Research*, pages 81–103.

[4] Balle, B., Quattoni, A., and Carreras, X. (2014). Spectral learning techniques for weighted automata, transducers, and grammars. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, Doha, Qatar. Association for Computational Linguistics.

[5] Casacuberta, F. and de la Higuera, C. (2000). Computational complexity of problems on probabilistic grammars and transducers. In Oliveira, A. L., editor, *Grammatical Inference: Algorithms and Applications*, pages 15–24. Springer Berlin Heidelberg.

[6] Cechin, A. L., Regina, D., Simon, P., and Stertz, K. (2003). State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings.*, pages 73–78.

[7] Chen, Y., Gilroy, S., Maletti, A., May, J., and Knight, K. (2018). Recurrent neural networks as weighted language recognizers. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2261–2271.

[8] Cortes, C., Mohri, M., and Rastogi, A. (2007). l_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(4):761–779.

[9] de la Higuera, C. (2010). *Grammatical inference: learning automata and grammars.* Cambridge University Press.

[10] de la Higuera, C., Scicluna, J., and Nederhof, M. (2014). On the computation of distances for probabilistic context-free grammars. arXiv preprint arXiv:, 2014.

[11] Elman, J. L. (1990). Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211.

[12] Giles, C. L., Chen, D., Miller, C. B., Chen, H. H., Sun, G. Z., and Lee, Y. C. (1991). Second-order recurrent neural networks for grammatical inference. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 273–281 vol.2.

[13] Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., and Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Comput.*, 4(3):393–405.

[14] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc.

[15] Karpathy, A., Johnson, J., and Li, F.-F. (2015). Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

[16] Kilian, J. and Siegelmann, H. T. (1993). On the power of sigmoid neural networks. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, page 137–143.

[17] Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press.

[18] Korsky, S. A. and Berwick, R. C. (2019). On the computational power of rnns.

[19] Lyngsø, R. B. and Pedersen, C. N. S. (2002). The consensus string problem and the complexity of comparing hidden markov models. *Journal of Computing and System Science*, 65(3):545–569.

[20] McCulloch, W. S. and Pitts, W. (1988). *A Logical Calculus of the Ideas Immanent in Nervous Activity*, page 15–27. MIT Press, Cambridge, MA, USA.

[21] Merrill, W. (2019). Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13, Florence. Association for Computational Linguistics.

[22] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. volume 2, pages 1045–1048.

[23] Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N. J.

[24] Okudono, T., Waga, M., Sekiyama, T., and Hasuo, I. (2019). Weighted automata extraction from recurrent neural networks via regression on state spaces.

[25] Oliva, C. and Lago-Fernandez, L. (2019). *On the Interpretation of Recurrent Neural Networks as Finite State Machines*, pages 312–323.

[26] Pollack, J. B. (1987). *On connectionist models of natural language processing*. PhD thesis, Computer Science Department, University of Illinois, Urbana.

[27] Rabin, M. O. (1963). Probabilistic automata. *Information and Control*, 6(3):230 – 245.

[28] Schellhammer, I., Diederich, J., Towsey, M., and Brugman, C. (1998). Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task. In *New Methods in Language Processing and Computational Natural Language Learning*.

[29] Siegelmann, H. and Sontag, E. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132 – 150.

[30] Thollard, F., de la Higuera, C., Vidal, E., Casacuberta, F., and Carrasco, R. C. (2005). Probabilistic finite-state machines-part i. *IEEE Transactions on Pattern Analysis $ Machine Intelligence*, 27(07):1013–1025.

[31] Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005). Probabilistic finite-state machines - part ii. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039.

[32] Watrous, R. L. and Kuhn, G. M. (1991). Induction of finite-state automata using second-order recurrent networks. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, page 309–316, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[33] Weiss, G., Goldberg, Y., and Yahav, E. (2018a). Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5247–5256.

[34] Weiss, G., Goldberg, Y., and Yahav, E. (2018b). On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th ACL*, pages 740–745. ACL.

[35] Zeng, Z., Goodman, R. M., and Smyth, P. (1993). Learning finite machines with self-clustering recurrent networks. *Neural Comput.*, 5(6):976–990.

[36] Zhang, Q., Wu, Y. N., and Zhu, S. (2018). Interpretable convolutional neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8827–8836.