entropy

MDPI

Article

An Entropy Metric for Regular Grammar Classification and Learning with Recurrent Neural Networks

Kaixuan Zhang [1], Qinglong Wang [2] and C. Lee Giles [1,*]

[1] Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802, USA; kuz22@psu.edu

[2] Alibaba Group, Building A2, Lane 55 Chuan He Road Zhangjiang, Pudong New District, Shanghai 200135, China; xifu.wql@alibaba-inc.com

* Correspondence: clg20@psu.edu

Abstract: Recently, there has been a resurgence of formal language theory in deep learning research. However, most research focused on the more practical problems of attempting to represent symbolic knowledge by machine learning. In contrast, there has been limited research on exploring the fundamental connection between them. To obtain a better understanding of the internal structures of regular grammars and their corresponding complexity, we focus on categorizing regular grammars by using both theoretical analysis and empirical evidence. Specifically, motivated by the concentric ring representation, we relaxed the original order information and introduced an entropy metric for describing the complexity of different regular grammars. Based on the entropy metric, we categorized regular grammars into three disjoint subclasses: the polynomial, exponential and proportional classes. In addition, several classification theorems are provided for different representations of regular grammars. Our analysis was validated by examining the process of learning grammars with multiple recurrent neural networks. Our results show that as expected more complex grammars are generally more difficult to learn.

Keywords: entropy; regular grammar classification; complexity analysis; recurrent neural network

check for updates

Citation: Zhang, K.; Wang, Q.; Giles, C. An Entropy Metric for Regular Grammar Classification and Learning with Recurrent Neural Networks. *Entropy* **2021**, *23*, 127. https://doi.org/10.3390/e23010127

Received: 5 December 2020
Accepted: 13 January 2021
Published: 19 January 2021

1. Introduction

Regular grammars (RGs) have been widely studied in the theory of computation and intensively applied in natural language processing, compiler construction, software design, parsing and formal verification [1–4]. Despite their importance and pervasiveness, there is limited research [5–7] investigating the internal structure of regular grammars. As such, our understanding of regular grammars is relatively coarse-grained.

One approach to understand a more fine-grained understanding of regular grammars is to investigate them through machine learning. Recent research has demonstrated that recurrent neural networks (RNNs) can achieve superior performance in a wide array of areas that involve sequential data [8], e.g., financial forecasting, language processing, program analysis and particularly grammatical inference. Specially, recent work has shown that certain types of regular grammars can be more easily learned by recurrent networks [9,10]. This is important in that it provides crucial insights in understanding regular grammar complexity. Furthermore, understanding the learning process of regular grammar also help differentiating different recurrent models [11–13].

Our intent is to establish a closer connection between the complexity of regular grammars and machine learning tasks from both theoretical and empirical perspectives. From a theoretical perspective, we regard classification and representation as two fundamental problems (Izrail Gelfand once remarked, "all of mathematics is some kind of representation theory" [14]). We follow previous work [15] by studying a grammar from its concentric ring graph representation, which contains sets of strings (with a certain length) accepted and rejected by this grammar. Note that this representation can be used for any grammar.

An entropy value is then introduced based on the properties of the concentric ring graph that categorizes all regular grammars into three classes with different levels of complexity, and further establishes several classification theorems based on different representations of regular grammars. In addition, through an empirical study, different regular grammars are categorized by applying them to a set of learning tasks. That is, given enough positive (accepted) and negative (rejected) string samples of a specific regular grammar, it is expected that machine learning models will gradually identify a latent pattern of a grammar through the training process. This shows that this indeed reflects the difficulty of learning regular grammars and highly depends on the complexity of a grammar. All RNNs were evaluated and compared on string sets generated by different RGs with different levels of complexity so as to explore the characteristics of each regular grammar. As such, the empirical results become well aligned with the theoretical analysis. It is hoped that these results can provide a deeper understanding of a regular grammar under a learning scenario as well as practical insights into its complexity.

In summary, the contributions of this paper are as follows:

- We propose a new entropy metric for measuring the complexity of regular grammar in terms of machine learning which reflects the internal structure and hence provides a fine-grained understanding of a regular grammar.
- We categorize regular grammars into three disjoint subclasses based on the entropy value with several classification theorems for different representations proved.
- We empirically demonstrate the validity of the proposed metric and corresponding classification by a series of experiments on the Tomita grammars.

The rest of the paper is organized as follows. Section 2 provides preliminary and background material on recurrent neural networks and regular grammars. Section 3 surveys relevant work on complexity and classification of regular grammars. Section 4 presents our complexity metric and categorization results with several theorems based on different representations proved. Case study and experimental results are shown in Sections 5 and 6, respectively. Section 7 is the conclusion and future work.

2. Preliminaries

First, there is a brief review of fundamental recurrent neural machine learning models. The definition of regular grammars is then introduced, followed by the set of Tomita grammars that are used in this study.

2.1. Recurrent Neural Networks

A unified view of the update activity of recurrent neurons for different RNNs investigated is shown in Table 1. Typically, a RNN consists of a hidden layer h containing N_h recurrent neurons (each designated as h_i) and an input layer x containing N_x input neurons (each designated as x_k). The values of h at tth and $t-1$th discrete times are denoted as h^t and h^{t-1}. Then, the hidden layer is updated by:

$$h^{t+1} = \phi(x^t, h^t, W), \tag{1}$$

where ϕ is the activation function (e.g., `Tanh` and `Relu`) and W denotes the weights which modify the strength of interaction among input neurons, hidden neurons, output neurons and any other auxiliary units. The hidden layer update for each RNN is presented in Table 1. Next, we describe fundamental RNN models used in our analysis.

2.1.1. Elman Network (SRN)

SRN [16] integrates the input layer and the previous hidden layer in a manner that is regarded as a "first-order" connection [17]. This first-order connection has been widely adopted for building different recurrent layers, for example the gate units in LSTM [18] and GRU [19].

Table 1. Hidden update of RNNs selected; building blocks for developing many complicated models. Let W_*, U_* and V_* denote weights designed for connecting different neurons and b denote the bias. $\odot$ is the Hadamard product.

Model	**Hidden Layer Update** $(\boldsymbol{U_* \in \mathbb{R}^{N_h \times N_x}, V_* \in \mathbb{R}^{N_h \times N_h}, b \in \mathbb{R}^{N_h \times 1}})$
SRN	$h^t = \phi(Ux^t + Vh^{t-1} + b)$
2-RNN	$h_i^t = \phi(\sum_{j,k} W_{kij} h_j^{t-1} x_k^t + b_i),\ i, j = 1, \cdots, N_h,\ k = 1, \cdots, N_x, W \in \mathbb{R}^{N_h \times N_h \times N_x}$
LSTM	$s^t = \phi(U_s x^t + V_s h^{t-1}), s = \{i, f, o, g\}$ and $\phi = \{\texttt{Sigmoid}, \texttt{Tanh}\}$ $c^t = c^{t-1} \odot f^t + g^t \odot i^t, h^t = \texttt{Tanh}(c_t) \odot o^t$
GRU	$z^t = \sigma(U_z x^t + V_z h^{t-1}), r^t = \sigma(U_r x^t + V_r h^{t-1}),$ $g^t = \texttt{Tanh}(U_h x^t + V_h(h^{t-1} \odot r^t)),$ $h^t = (1 - z^t) \odot g^t + z^t \odot h^{t-1}$

2.1.2. Second-Order RNNs (2-RNN)

2-RNN [20] have second-order connections in their recurrent layers and are designed to capture more complex interactions between neurons. The 2-RNN has a recurrent layer updated by a weighed product of input and hidden neurons. This type of connection enables a direct mapping between 2-RNN and a DFA [21]. Recent work [22] also shows the equivalence between a 2-RNN with linear hidden activation and weighted automata. Related work [23,24] used weighted automata to explore the relationships between deep learning and grammatical inference. Since a 2-RNN has a 3D tensor weight, computation is more intensive. As such, various approximations (M-RNN with a tensor decomposition [25] and MI-RNN with a rank-1 approximation [26]) have been proposed to alleviate the computational cost while preserving the benefits of high order connections for better modeling the complicated recurrent interaction.

2.1.3. RNNs with Gated Units

LSTM [18] and GRU [19] were proposed to deal with the vanishing and exploding gradient problems of SRNs. While these RNNs are effective for capturing the long-term dependence between sequential inputs, their gate units induce highly nonlinear behavior to the update of the hidden layer which creates difficulty in analysis.

2.2. Regular Grammar

A regular grammar (RG) recognizes and generates a regular language—a set of strings of symbols from an alphabet—and is uniquely associated with a deterministic finite automata (DFA) with a minimal number of states [27]. For a more thorough and detailed description of regular language and finite state machines, please refer to the work of Hopcroft et al. [27].

The Tomita grammars [28] denote a set of seven regular grammars that have been widely adopted in the study of grammatical inference [29]. Even though the Tomita grammars are relatively simple, they cover regular languages that have a wide range of complexities. and have been widely used as benchmarks [30–32] for grammar learning. These grammars all have alphabet $\Sigma = \{0, 1\}$ and generate infinite languages over $\{0, 1\}^*$. For each Tomita grammar, the binary strings generated by this grammar are its associated positive samples and the rest of the binary strings as negative samples. A description of the positive samples accepted by all Tomita grammars is shown in Table 2. It is important to realize that DFAs cover a wide range of languages, which means that all languages whose string length and alphabet size are bounded can be recognized and generated by a DFA [20]. It is worth noting that regular grammars and finite state machines have many practical uses in mechanical and electrical engineering and software.

Table 2. Descriptions of the Tomita grammars. The Kleene star is represented by $*$.

G	Description
1	1 *
2	(10)*
3	an odd number of consecutive 1 s are always followed by an even number of consecutive 0 s
4	any string not containing "000" as a substring
5	even number of 0 s and even number of 1 s
6	the difference between the number of 0 s and the number of 1 s is a multiple of 3
7	0*1*0*1*

3. Related Work

Here, we briefly discuss RNNs and regular grammars. We also revisit several complexity measure and some traditional representations of the regular grammar in automata theory.

3.1. RNNs and Regular Grammars

Grammars and automata and neural networks have been intertwined since McCulloch and Pitts' early 1943 paper [33], which led to Kleene's work [34] on regular grammars. Minsky's dissertation [35] extended this to models of neural networks and automata. With the rebirth of neural networks, much work on recurrent networks and finite state automata [16,20,36,37] was restarted. Since then, there has been theoretical work on Turing equivalence [38] and finite state machine encoding and representation [39]. The computational hardness of some of the representation results have recently been discussed [12]. Recently, there has been a renewed interest in extracting learned DFA from recurrent neural networks [31,40,41]. Such methods can be important for verification for neural networks and explainable AI [4,42] There has been an increase in the natural language processing community on revisiting formal language models [43]. Because of their wide use and applications, we only focus on regular grammars.

3.2. Complexity of Regular Grammars

3.2.1. Complexity of Shift Space

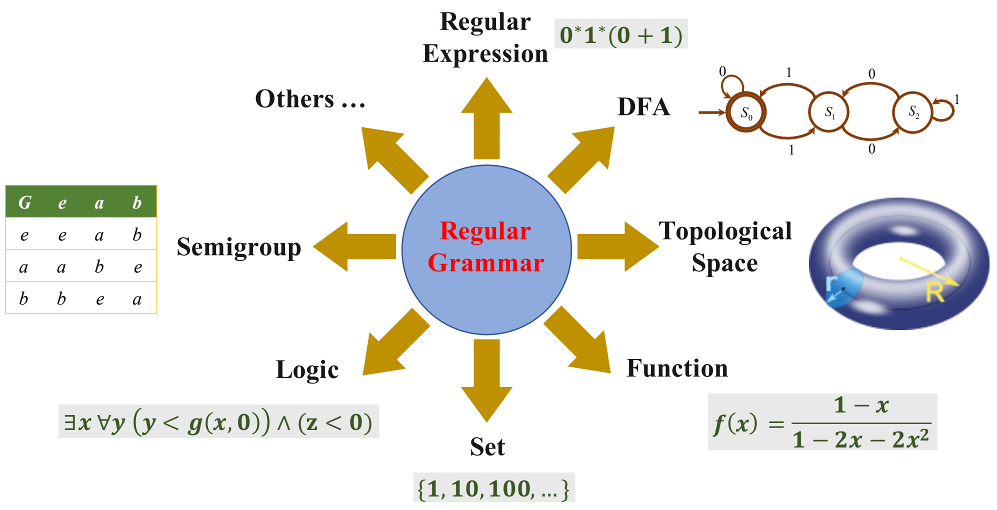
In symbolic dynamics [44], a particular form of entropy is defined to measure the "information capacity" of the *shift space*, which is a set of bi-infinite symbolic sequences that represent the evolution of a discrete system. When applied to measure the complexity of a RG, this entropy describes the cardinality of the strings defined by its language.

3.2.2. Logical Complexity

RG can also be categorized according to logical complexity [45]: strictly local (SL), strictly piecewise (SP), locally testable (LT), etc. These classes have multiple characterizations in terms of logic, automata, regular expressions and abstract algebra [6]. SL and SP languages are the simplest and most commonly used languages that define a finite set of factors and subsequences, respectively, and are selected to evaluate different RNNs on their performance in capturing the long-term dependency [6]. Unlike logical categorizations of regular grammars, we evaluate the complexity in terms of machine learning tasks.

3.3. Representations of Regular Grammars

Automata theory has a close connection to many different mathematical subjects such as symbolic dynamics, group theory, algebraic geometry, topology and mathematical logic [46]. Hence, a regular grammar has different representations based on these perspectives. A description of several commonly used representations is shown in Figure 1. We introduce the necessary concepts when they are used in the paper.



G	e	a	b
e	e	a	b
a	a	b	e
b	b	e	a

Figure 1. Different representations of regular grammar.

3.4. Entropy and Regular Grammars

It is important to note that the concept of entropy was introduced in the field of grammatical inference to solve different tasks [47,48] and more recently for distilling weighted automata [49]. Unlike those definitions, our definition originates in the graphical representation of regular grammar and gives insight into a metric for evaluating RNN learning and DFA extraction.

4. Categorization of Regular Grammars

This section reviews the concentric ring representation of regular grammar, and, using this representation, we introduce an entropy metric to evaluate their complexity. All RGs are then categorized into three classes according to their entropy values. Last, we provide several classification theorems of RGs in terms of their different representations. A flowchart of our analysis is shown Figure 2.

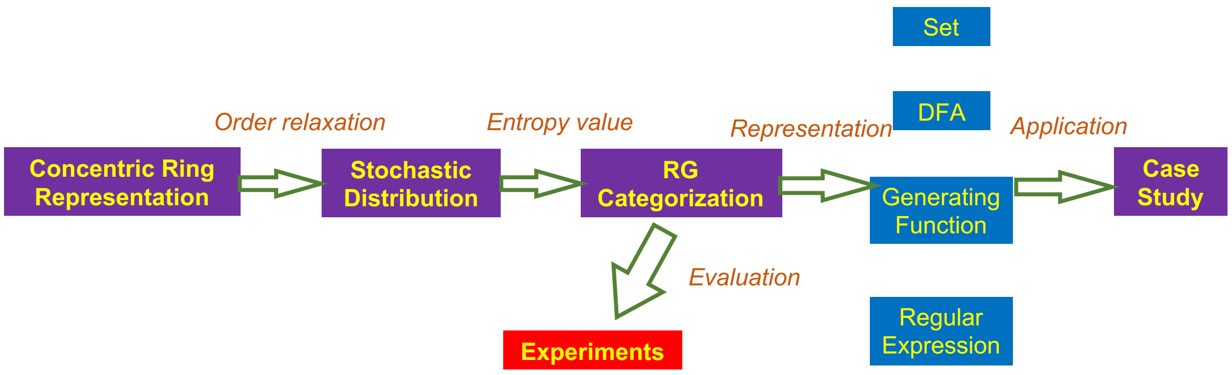


Figure 2. Flowchart of analysis.

4.1. Entropy of a Regular Language from an Concentric Ring Representation

The concentric ring representation [15] of a regular grammar reflects the distribution of its associated positive and negative strings within a certain length. Specifically, in each concentric ring, all strings with the same length are arranged in lexicographic order where white and black areas represent accepted and rejected strings respectively. Three ring graphs for Tomita Grammars 1, 3 and 6 are shown in Figure 3 to illustrate their differences. In each graph, every concentric ring contains the sets of strings and its following strings at a specific length that are accepted and rejected by its RG. Note that the percentages of accepted (or rejected) strings for different grammars are very different. For example, Grammars 3 and 6 have the numbers of accepted strings much larger than that of Grammar 1. This difference in prior empirical work [31,41] showed that Grammar 6 is much harder to learn than Grammars 1 and 3. An intuitive explanation is that, for Grammar 6, flipping any 0 to 1 or vice versa means any accepted or rejected string can be converted into a string with the opposite label. A RNN needs to learn such subtle changes in order to correctly recognize all strings accepted by Grammar 6. Since this change can happen to any digit, a RNN must account for all digits without neglecting any.

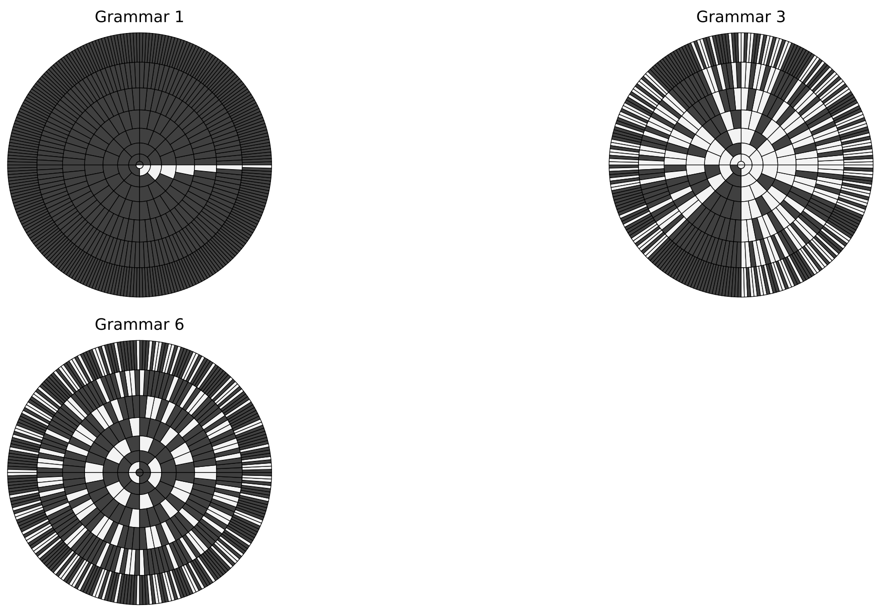


Figure 3. Concentric ring representation of the distribution of strings of length N ($1 \leq N \leq 8$) for Grammars 1, 3 and 6. Each concentric ring of a graph has 2^N strings arranged in lexicographic order, starting at $\theta = 0$. (See all graphs in Appendix A.)

We now formally show that a RG that generates a more balanced set of accepted and rejected strings has a higher level of complexity and appears more difficult to learn. Given an alphabet $\Sigma = \{0, 1\}$, the collection of all 2^N strings of symbols from Σ with length N is denoted as X^N. For a grammar G, let m_p^N (r_p^N) and m_n^N (r_n^N) be the numbers (ratios) of positive and negative strings, respectively. The constraint that all strings are arranged in a lexicographic order is relaxed, which indicates that all strings in X^N are randomly distributed. We then denote the expected times of occurrence for an event F_N—two consecutive strings having different labels—by $\mathrm{E}[F_N]$. This gives the following definition of entropy for RGs with a binary alphabet.

Definition 1 (Entropy)**.** *Given a grammar G with alphabet $\Sigma = \{0, 1\}$, its entropy is:*

$$H(G) = \limsup_{N \to \infty} H^N(G) = \limsup_{N \to \infty} \frac{1}{N} \log_2 \mathrm{E}[F_N], \tag{2}$$

where $H^N(G)$ is the entropy calculated for strings with the length of N. (Here, we use lim sup *for certain particular cases, i.e., when N is set to an odd value for Grammar 5.) .*

Furthermore, the following proposition to efficiently calculate the entropy by expressing $\mathrm{E}[F_N]$ explicitly becomes:

Proposition 1.

$$H(G) = 1 + \limsup_{N \to \infty} \frac{\log_2 \left(r_p^N \cdot r_n^N \right)}{N}. \tag{3}$$

Proof of Proposition 1. Given any concentric ring (corresponding to the set of strings with a length of N) shown in Figure 3, let R denote the number of consecutive runs of strings and R_p and R_n denote the number of consecutive runs of positive strings and negative strings in this concentric ring respectively. Then, we have $\mathrm{E}[F] = \mathrm{E}[R] - 1 = \mathrm{E}[R_p] + \mathrm{E}[R_n] - 1$. Without loss of generality, we can choose the first position as $\theta = 0$ in the concentric ring. Then, we introduce an indicator function I by $I_i = 1$ representing that a run of positive strings starts at the ith position and $I_i = 0$ otherwise. Since $R_p = \sum_{i=1}^{2^N} I_i$, we have

$$\mathrm{E}[R_p] = \sum_{i=1}^{2^N} \mathrm{E}[I_i] \quad \text{and} \quad \mathrm{E}[I_i] = \begin{cases} m_p/2^N, & i = 1 \\ m_n m_p/2^N(2^N-1), & i \neq 1. \end{cases}$$

As such, we have

$$\mathrm{E}[R_p] = \frac{m_p(1+m_n)}{2^N} \quad \text{and} \quad \mathrm{E}[R_n] = \frac{m_n(1+m_p)}{2^N}.$$

By substituting $\mathrm{E}[F]$ into the entropy definition, we have

$$H(G) = 1 + \limsup_{N\to\infty} \frac{\log_2 (r_p^N \cdot r_n^N)}{N}. \tag{4}$$

□

Thus, the metric entropy is well-defined and lies between 0 and 1. Proposition 1 implies that the entropy of an RG is equal to the entropy of its complement, which confirms our intuition in terms of learning task. Without loss of generality, we assume from now on that for all RGs the set of accepted strings has a smaller cardinality. In addition, we conclude that an RG generating more a balanced string sets has a higher entropy value. As such, we can categorize all RGs with a binary alphabet based on their entropy values.

Definition 2 (Subclass of Regular Grammar)**.** *Given any regular grammar G with $\Sigma = \{0,1\}$, we have:*

(a) G belongs to ***Polynomial*** *class if $H(G) = 0$;*
(b) G belongs to ***Exponential*** *class if $H(G) \in (0,1)$; and*
(c) G belongs to ***Proportional*** *class if $H(G) = 1$.*

Formally speaking, the metric entropy defines an equivalence relation, denoted as $\sim_H$. A subclass of regular grammar can be considered as the equivalence class by the quotient $\mathbf{RG}/\sim_H$, where **RG** denotes the sets of all RGs. In this way, we have $\mathbf{RG}/\sim_H = \{[\mathbf{Po}], [\mathbf{Ex}], [\mathbf{Pr}]\}$, where $[\mathbf{Po}], [\mathbf{Ex}], [\mathbf{Pr}]$ denote polynomial, exponential, and proportional classes, respectively.

When compared to the entropy in shift space, which only considers accepted strings, our Definition 1 considers both the accepted and rejected strings. This is more informative and has other benefits. For example, given a dataset with samples uniformly sampled from an unknown dataset, we can then estimate the complexity of this unknown dataset by calculating the entropy of the available dataset. For a k-class classification task with strings of length N, let m_i denote the number of strings in the ith class. Then, we have $\mathrm{E}[F_N] = 2^N - \frac{1}{2^N} \cdot \sum_{i=1}^{k} m_i^2$. We can then generalize Definition 1 to a k-class classification case by substituting this into Definition 1. However, this can be challenging for the entropy defined for shift space since it can only be constructed in a one-versus-all manner. In addition, the shift space cannot express all RGs, especially for grammars that lack shift-invariant and closure properties [44].

4.2. Set Representation

Here, we conduct a combinatorial analysis. Given a regular grammar G, we consider it as a set and explore the property of its cardinality. Namely, we have the following theorem:

Theorem 1. *Given any regular grammar G with $\Sigma = \{0,1\}$, we have:*

(a) G belongs to $[\mathbf{Po}]$ if and only if $m_p^N \sim P(N)$, where $P(N)$ denotes the polynomial function of N;

(b) G belongs to $[\mathbf{Ex}]$ if and only if $m_p^N \sim \beta \cdot b^N$ where $b < 2$ and $\beta > 0$ and $H(G) = \log_2 b$; and

(c) G belongs to $[\mathbf{Pr}]$ if and only if $m_p^N \sim \alpha \cdot 2^N$, where $\alpha \in [0,1)$.

Here, $\sim$ indicates that some negligible terms are omitted when N approaches infinity.

Proof of Theorem 1. In both Definition 2 and Proposition 1, $\limsup$ is used to cover certain particular cases, for instance when N is set to odd value for Grammar 5. In the following proof, without loss of generality, $\lim$ is used instead of $\limsup$ for simplicity. According to Proposition 1, for any regular grammar G, its entropy $H(G) \in [0,1]$. It can be checked that the maximum value of $H(G)$ is 1 when $r_p^N = 0.5$. In addition, the minimum value of $H(G)$ is 0 and can be reached when $r_p^N = 0$ or 1. However, $r_p^N = 0$ or 1 are only allowed for grammars that either accept or reject any string, hence are not considered in this theorem. As such, in this case, the value of entropy is taken as the minimum when $r_p^N = 1/2^N$ or $1 - 1/2^N$. In the following, we only discuss the former case and the latter can be similarly derived.

For each class of grammars, given that their m_p takes the corresponding form shown in Theorem 1, the proof for the sufficient condition is trivial and can be checked by applying *L'Hospital's Rule*. As such, in the following, we only provide a proof for the necessary condition.

From (4), we have:

$$
\begin{aligned}
H(G) &= \lim_{N\to\infty} \frac{\log_2(m_p \cdot 2^N - m_p^2)}{N} - 1 \\
&= \lim_{N\to\infty} \frac{m_p' \cdot 2^N + \ln 2 \cdot 2^N \cdot m_p - 2m_p \cdot m_p'}{\ln 2 \cdot (m_p \cdot 2^N - m_p^2)} - 1 \\
&= \lim_{N\to\infty} \frac{m_p' \cdot 2^N + \ln 2 \cdot m_p^2 - 2m_p \cdot m_p'}{\ln 2 \cdot m_p \cdot (2^N - m_p)},
\end{aligned}
$$

where m_p' denotes the derivative of m_p with respect to N. It is easy to check that $\lim_{N\to\infty} \frac{m_p'}{m_p}$ exists for regular grammars. Then, we separate the above equation as follows:

$$
H(G) = \lim_{N\to\infty} \frac{m_p'}{\ln 2 \cdot m_p} + \lim_{N\to\infty} \frac{1 - \frac{m_p'}{\ln 2 \cdot m_p}}{\frac{2^N}{m_p} - 1}.
$$

It should be noted that the second term in the above equation equals 0. Specifically, assuming that m_p has the form of $\alpha \cdot b^N$ where $b < 2$ (b cannot be larger than 2 for binary alphabet), then the denominator of the second term is infinity. If m_p has the form of $\alpha \cdot 2^N$, then the numerator tends to zero while the denominator is finite. As such, we have

$$
H(G) = \lim_{N\to\infty} \frac{m_p'}{\ln 2 \cdot m_p}.
$$

If $H(G) = 0$, then we have $\lim_{N\to\infty} \frac{m_p'}{m_p} = 0$, indicating that the dominant part of m_p has a polynomial form of N hence $m_p \sim P(N)$, where $P(N)$ denotes the polynomial function of N.

If $H(G) = t \neq 0$, then we have $\lim_{N\to\infty} \frac{\ln(m_p)}{tN\ln 2} = 1$, which gives that $m_p \sim \beta \cdot 2^{tN}$, where $\beta > 0$. If $t = \log_2 b$, then we have $m_p \sim \beta \cdot b^N$ where $b < 2$. Furthermore, if $t = 1$, we have $m_p \sim \alpha \cdot 2^N$ where $\alpha \in [0, 1)$. $\square$

Theorem 1 shows the reason for naming these subclasses. In addition, when a specific order is posed on the set of given RG, we have a similar result from number theory. Specifically, given a string $s = x_1x_2x_3 \cdots x_N$ with length N, we associate the string with a decimal number, i.e., $D_s = \sum_{i=1}^{N} x_i \cdot 2^{i-1} + 2^N - 1$. Note that this formula is different from the traditional approach to transforming a binary number to a decimal number, since 0 has physical meaning in regular grammar. For example, 000 is different from 00 in regular grammar. That is the reason that we need an additional term term $2^N - 1$ to differentiate these situations. In this way, D_s has induced an order on the regular grammar. Let D_s^n denote the nth smallest number in this set; then, we have the following result:

Corollary 1. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) *G belongs to* $[\mathbf{Po}]$ *if and only if* $\lim_{n\to\infty} B_n = 0$;
(b) *G belongs to* $[\mathbf{Ex}]$ *if and only if* $\lim_{n\to\infty} B_n \in (0, 1)$; *and*
(c) *G belongs to* $[\mathbf{Pr}]$ *if and only if* $\lim_{n\to\infty} B_n = 1$.

Here, $B_n = log_2 n / \log_2 D_s^n$.

Here, we provide a brief explanation of Corollary 1. The denominator approximates the length of the string and the numerator represents the cardinality of the set. Hence, comparing with the original Definition 2, this corollary provide an alternative perceptive of entropy. Take Tomita Grammar 1 for example. The set is $G = \{\epsilon, 1, 11, 111, \cdots\}$ with the number $D_s = \{0, 2, 6, 14, \cdots\}$, and the formula for D_s^n is given by $D_s^n = 2^n - 2$; hence, by simple calculation, we find that the limit of the ratio is 0 when n approaches to infinity, which implies that Grammar 1 belongs to the polynomial class. Note that in general it is difficult to calculate the explicit formula for D_s^n and therefore Corollary 1 has certain limitation in practical applications.

4.3. DFA Representation

Here, we provide an alternative way to obtain the classifications for RGs using the transition matrices of its associated minimal DFA [27] in terms of states. This approach provides immediate results if the minimal DFA is available. As such, this reduces the computation cost of a data-driven approach. Here, we again use the case when the alphabet size is two. However, it is easy to extend this for grammars with larger alphabets. Given a regular grammar G with the alphabet $\Sigma = \{0, 1\}$ and its associated minimal complete DFA M with n states, let T_0, $T_1 \in \mathbb{Z}^{n\times n}$ denote the transition matrices of M associated with input 0 and 1, and the transition matrix of the DFA is defined as $T = T_0 + T_1$. Alternatively, the transition matrix of a given DFA can be considered as the transition matrix of underlying directed graph by neglecting the labels associated with the edges. With the above settings, we have the following theorem:

Theorem 2. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) *G belongs to* $[\mathbf{Po}]$ *if and only if* $k(T) = 1$ *and* $\sigma(T) = \{1, 2\}$;
(b) *G belongs to* $[\mathbf{Ex}]$ *if and only if* $k(T) = 1$ *and* $\sigma(T) - \{1, 2\} \neq \emptyset$, *and* $H(G) = \log_2|\lambda_2|$ *where* $|\lambda_2|$ *denotes the second largest modulus of the eigenvalues of T; and*
(c) *G belongs to* $[\mathbf{Pr}]$ *if and only if* $k(T) = 0$ *or* $k(T) = 2$.

Here, $k(T)$ represents the number of diagonal elements equal to 2 and $\sigma(T)$ denotes the set of modulus of all eigenvalues of T.

Proof of Theorem 2. We first introduce a lemma which is used in the proof.

Lemma 1. *Let v_1 and v_2 denote two one-hot encoded column vectors corresponding to a starting state and an ending state of a DFA respectively. One can construct an adjacent matrix P for this DFA by regarding it as an undirected graph with each node represents a state and every edge represents the existence of a transition between a pair of states. Then, the number of L-length strings that reach the ending state v_2 from v_1 is $v_1^\mathrm{T} P^N v_2$.*

Based on the lemma above, it is easy to see that the number of positive strings is $m_p = v_1^\mathrm{T} P^N q$ where q is a one-hot encoded column vector representing an accepting state. By applying Jordan decomposition to P, i.e., $P = SJS^{-1}$, we can see that $m_p = v_1^\mathrm{T} S J^N S^{-1} q$ and J^N is the only term depending on N. Specifically, take one Jordan block $J_i \in \mathbb{R}^{m\times m}$ with the eigenvalue of λ and let K_i denote the nilpotent matrix associated with J_i, that is, the superdiagonal of K_i contains ones and all other entries are zero. Then, we have:

$$J_i(\lambda)^N = (\lambda I + K_i)^N = \sum_{n=0}^{\min(N,m-1)} \binom{N}{n} \lambda^{N-n} K_i^n.$$

It is easy to see that when the absolute of the eigenvalue, i.e., $|\lambda| = 1$, $J_i(\lambda)^N$ has a polynomial form of N. This result can be generalized to all Jordan blocks of J. As shown in the proof of Theorem 1, this corresponds to the case when G belongs to the polynomial class (we omit the proof since the number of diagonal elements of T equal to 2 is 1, as discussed in the paper). Denote the second largest eigenvalue of T as b, then $\beta \cdot b^N$ dominates m_p, where β is some constant. As such, one can easily derive the proof by following the proof for the exponential class in Theorem 1. □

Theorem 2 indicates that the entropy of a RG lies in the spectrum of its associated DFA. Specifically, in the polynomial and exponential classes, a DFA with its summed transition matrix T having only one diagonal element that is equal to 2 indicates that this DFA has only one "absorbing" state (either the accepting state or the rejecting state). Assume that a DFA has one absorbing state and is running over a string. Once reaching the absorbing state, this DFA makes a permanent decision—either acceptance or rejection—on this string, regardless of the ending symbol has been read or not. In comparison, in the proportional class, a DFA can have either zero or two absorbing states (one accepting state and one rejecting state). In the case of Tomita grammars, every grammar has exactly one absorbing state except for Grammars 5 and 6, which have no absorbing states. The DFAs for Grammars 5 and 6 can only determine the label of a string after processing the entire string.

Recall that we require that for all RGs the set of accepted strings has a smaller cardinality. As such, the incomplete DFA for a given RG can be obtained by deleting the rejecting absorbing state and all the edges associated with the state from a minimal complete DFA. (This is important since for a proportional class there might exist two absorbing states and the accepting absorbing state in that case cannot be deleted. A more significant result is that, in the monoid representation of regular grammar, only the monoid associated with polynomial and exponential grammar is a nulloid. The discussion of monoid representation is out scope here.) Similarly, the transition matrix $\hat{T}$ of the incomplete DFA becomes:

Corollary 2. *Given any regular grammar G with $\Sigma = \{0,1\}$, we have:*

(a) *G belongs to $[\mathbf{Po}]$ if and only if $L(\hat{T}) = 1$;*

(b) *G belongs to $[\mathbf{Ex}]$ if and only if $L(\hat{T}) \in (1,2)$ and $H(G) = \log_2 |L(\hat{T})|$; and*

(c) *G belongs to $[\mathbf{Pr}]$ if and only if $L(\hat{T}) = 2$.*

Here, $L(\hat{T})$ denotes the largest eigenvalue of $\hat{T}$.

Corollary 2 together with Theorem 2 provides a complete analysis of classification results for a DFA representation. It is easy to see that neither strictly local or strictly piecewise belongs to the proportional class since they all have one absorbing-rejecting state. They can be categorized into either the polynomial or exponential class according to their specific forbidden factors or subsequences. Please refer to classification theorems of regular grammars with arbitrary alphabet size in Appendix B.

4.4. Generating Function Representation

Now, we derive the classification results based on the generating function representation of a regular grammar. A generating function encodes an infinite sequence of numbers by treating them as the coefficients of a power series, which is widely applied in combinatorial enumeration. Specifically, the generating function of regular grammar is defined as follows:

$$f(x) = \sum_N m_p^N \cdot x^N, \tag{5}$$

An important result is that generating function of regular grammar is a rational function. We have the following theorem for classification:

Theorem 3. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) G belongs to [**Po**] *if and only if $\{m_p^N\}$ is finite or the radius of convergence r of $f(x)$ is equal to 1;*
(b) G belongs to [**Ex**] *if and only if the radius of convergence r of $f(x)$ is between 1/2 and 1, and $H(G) = -\log_2 r$; and*
(c) G belongs to [**Pr**] *if and only if the radius of convergence r of $f(x)$ is equal to 1/2.*

The theorem can be understood from two perspectives. First, from Theorem 1, we can readily derive the radius of convergence by a ratio test. Another is related to calculating the generating function from a DFA. Specially, we have the following lemma [50]:

Lemma 2. *The generating function $f_{ij}(x)$ from state i to state j is given by*

$$f_{ij}(x) = \frac{(-1)^{i+j}\det(I - xT : j, i)}{\det(I - xT)}, \tag{6}$$

where $(B : j, i)$ denotes the matrix obtained by removing the jth row and ith column of B.

Note that the radius of convergence r only depends on the denominator of the function $f(x)$, and r is the smallest pole of the function. By Lemma 2, the denominator has the form $\det(I - xT)$. As such, the radius is the inverse of the largest eigenvalue of the transition matrix. The classification theorem is a generating function of a regular grammar and can be easily generalized to a regular grammar with multiple alphabets.

4.5. Regular Expression Representation

Here, we provide an analysis from a regular expression perspective. First, we consider the following interesting question: Given a regular grammar, how many parameters does one need to uniquely determine a positive string? The question is closely related to automatically generate positive samples by computer. We start with a simple example. For Tomita Grammar 7, we have $G = 0^*1^*0^*1^*$, and we only need four numbers a, b, c, d to generate a string $0^a1^b0^c1^d$. However, for a more complicated example Tomita Grammar 4, we have $G = (\epsilon + 0 + 00)(1 + 10 + 100)^*$, and, in this case, we have to record both the number and location of the suffix string 1, 10, 100 separately. Hence, we need $\mathbb{Z}^3$ numbers to generate a sample. Moreover, we have the following fact:

Fact 1. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) G belongs to [**Po**] *if and only if $N(G) = k$;*
(b) G belongs to [**Ex**] *if and only if $N(G) = \mathbb{Z}^k$; and*
(c) G belongs to [**Pr**] *if and only if $N(G) = \mathbb{Z}^{\mathbb{Z}^k}$.*

Here, k is a constant and $N(G)$ denotes the required number of parameters.

Another perspective to explore this problem is to apply group theory. Specifically, given a regular grammar, we can consider it as a topological space. Furthermore, it can be decomposed into several disjoint orbits of different base points, and the whole topological space is generated by the actions on these base points. In this way, these actions form a monoid and the cardinality of the monoid reflects the complexity of the grammar. Again, we use Tomita Grammars 4 and 7 as an example to illustrate this idea. For $G = 0^*1^*0^*1^*$, the base point of the topological space is ϵ and we define the following different actions: h_1, adding 0 in the first slot; h_2, adding 1 in the second slot; h_3, adding 0 in the third slot; and h_4, adding 1 in the fourth slot. Note that first adding 1 in the second slot and then adding 0 in the first slot is exactly the same as first adding 0 in the first slot and then adding 1 in the second slot, which means that these actions commute to each other. Hence, this monoid is an abelian monoid (more formally, this monoid can be considered as the quotient of a free monoid by a normal submonoid, i.e., $H = F\{h_1, h_2, h_3, h_4\}/ < h_i h_j h_j^{-1} h_i^{-1} >, \forall i \neq j$) and the cardinality of the monoid is $\mathbb{Z}^4$. For $G = (\epsilon + 0 + 00)(1 + 10 + 100)^*$, the base points are ϵ, 0 and 00, and we define the following actions: h_1, attaching 1 in the back; h_2, attaching 10 in the back; and h_1, attaching 100 in the back. Note that this monoid H is no longer commutative, since $h_1 h_2$ acting on base point 0 gives the string 0101 while $h_2 h_1$ acting on base point 0 gives the string 0011. Hence, in this case, the monoid is a free monoid generated by h_1, h_2, h_3 and the cardinality of this monoid is $3^{\mathbb{Z}}$. More generally, we have the following fact:

Fact 2. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) *G belongs to [**Po**] if and only if $C(H) = k$ or $C(H) = \mathbb{Z}^k$;*

(b) *G belongs to [**Ex**] if and only if $C(H) = k^{\mathbb{Z}}$; and*

(c) *G belongs to [**Pr**] if and only if $C(H) = k^{\mathbb{Z}^{\mathbb{Z}}}$.*

Here, k is a constant and $C(H)$ denotes the cardinality of the monoid of actions on the topological space.

Facts 1 and 2 give the following theorem:

Theorem 4. *Given any regular grammar G with $\Sigma = \{0, 1\}$, we have:*

(a) *G belongs to [**Po**] if and only if there is no $+$ inside the Kleene star $*$ in the regular expression;*

(b) *G belongs to [**Ex**] if and only if there exists a polynomial grammar inside the Kleene star $*$ in the regular expression; and*

(c) *G belongs to [**Pr**] if and only if there exists an exponential grammar inside the Kleene star $*$ or $(0+1)^{k*}$ in the regular expression, where k is a constant.*

Theorem 4 provides an analysis based on the regular expression. Note that, except for the special case in proportional class, this result does not depend on the size of alphabet. More examples are presented in the next section.

5. Case Study for Tomita Grammars

As our goal is to study the characteristics of subclasses of regular grammar in terms of machine learning, we use the often cited Tomita grammars [51] as examples for our analysis, which is consistent with the evaluation. For each classification result in Section 4, we choose an example to illustrate its application. The full results for the Tomita grammars are listed in Table 3.

Table 3. Analysis of Tomita grammars. ($\beta = 1/3 \cdot (19 + 3\sqrt{33})^{1/3} + 1/3 \cdot (19 - 3\sqrt{33})^{1/3} + 1/3$ and $b = \{3(586 + 102\sqrt{33})^{1/3}\}/\{(586 + 102\sqrt{33})^{2/3} + 4 - 2(586 + 102\sqrt{33})^{1/3}\}$. This is also known as a *Tribonacci number*.)

G	m_p^N	$L(\hat{T})$	$f(x)$	**Regular Expression**	**Class**
1	1	1	$\frac{1}{1-x}$	1^*	Po
2	$0.5 + 0.5(-1)^N$	1	$\frac{1}{1-x^2}$	$(10)^*$	Po
3	-	1.77	$\frac{(1+x)^2}{1-2x^2-2x^3}$	$0^*((11)^+0^* +$ $1(11)^*(00)^+)^*(\epsilon + 1(11)^*)$	Ex
4	$\beta \cdot b^{N\ 1}$	1.84	$\frac{1+x+x^2}{1-x-x^2-x^3}$	$(\epsilon + 0 + 00)(1 + 10 + 100)^*$	Ex
5	$2^{N-2}(1 + (-1)^N)$	2	$\frac{1-2x^2}{1-4x^2}$	$(00 + 11)^*((01 + 10)(00 + 11)^*$ $(01 + 10)(00 + 11)^*)^*$	Pr
6	$(2^N - 2(-1)^N)/3$	2	$\frac{1-x}{(1-2x)(1+x)}$	$(10 + 01)^*((1 + 00)(01 + 10)^*$ $(11 + 0)(01 + 10)^* + (11 + 0)(01 + 10)^*$ $(1 + 00)(01 + 10)^*)^*$	Pr
7	$(N^3 + 5N + 6)/6$	1	$\frac{1-2x+2x^2}{(1-x)^4}$	$0^*1^*0^*1^*$	Po

5.1. Set Representation

Here, we use Tomita Grammar 3 to illustrate the application of Theorem 1. Note that, in Table 3, we can see that it is difficult to derive an explicit formula for m_p^N from a pure combinatorial approach. Instead, we provide a brief proof that Grammar 3 belongs to exponential class. Let G^N denote the set of positive samples with length N, and it is easy to notice that it can be decomposed into the following four disjoint sets: samples ending with consecutive even numbers of 1 s, samples ending with consecutive odd numbers of 1 s, samples ending with 0 s and the number of 0 s at the end limited by the constraint, and samples ending with 0 s and the number of 0 s at the end not limited by the constraint. We use A_1^N, A_2^N, A_3^N,and A_4^N to denote these sets, respectively. Hence, we have that $m_p^N = A_1^N + A_2^N + A_3^N + A_4^N$. Furthermore, we have the following recursion formulas:

$$\begin{aligned} A_1^N &= A_2^{N-1}, \\ A_2^N &= A_1^{N-1} + A_3^{N-1} + A_4^{N-1}, \\ A_3^N &= \sum_{i=1}^{[(N-1)/2]} A_2^{N-2i}, \\ A_4^N &= A_4^{N-1} + A_1^{N-1}, \end{aligned} \tag{7}$$

where $[\cdot]$ denotes the floor function and the initial condition is $A_1^0 = 0$, $A_2^0 = 1$, $A_3^0 = 0$, and $A_4^0 = 1$. By adding these formulas, we have $m_p^N = m_p^{N-1} + m_p^{N-2} + A_2^{N-2}$, which implies that this sequence increases faster than Fibonacci sequence. Hence, it cannot belong to the polynomial class.

In contrast, let Z_1 denote the event that an even number of 0 s followed by odd number of 1 s and Z_2 denotes the event that even number of 0 s always followed by odd number of 1 s in the whole string. Hence, when the length of the string N approaches infinity, the expectation of the probability of Z_2 is an infinite product of probability of Z_1, which approaches zero. The indicates that the ratio of positive samples approaches zero when N approaches infinity. Thus, m_p^N does not contain the term $\alpha \cdot 2^N$. Finally, we conclude that this grammar belongs to exponential class.

5.2. DFA Representation

Here, we use Tomita Grammar 2 to illustrate the application of Theorem 2. The transition matrix of the complete DFA is $T = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 2 \end{bmatrix}$ obtained from Figure 4.

By simple calculation, we have $k(T) = 1$ and $\sigma(T) = \{1,2\}$. In addition, the transition matrix of the incomplete DFA is $\hat{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, and the largest eigenvalue is 1, which means it belongs to the polynomial class.

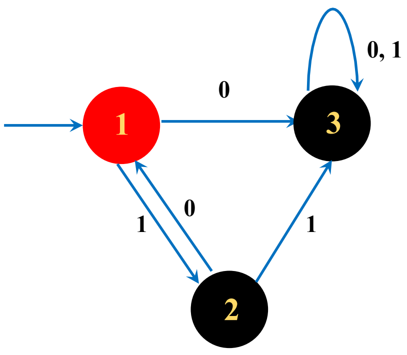


Figure 4. Example for the Tomita Grammar 2 where red (black) states are the accept (reject) states.

5.3. Generating Function Representation

Here, we use Tomita Grammar 4 to illustrate the application of Theorem 3. The generating function is $f(x) = \frac{1+x+x^2}{1-x-x^2-x^3}$, and the radius of convergence is the smallest positive pole of the function. Namely, we need to solve the equation $x^3 + x^2 + x - 1 = 0$ and we have $r = 0.544$, which lies between the values of 0 and 1. Hence, it belongs to the exponential class.

5.4. Regular Expression Representation

Here, we use Tomita Grammar 5 to illustrate the application of Theorem 4. For $G = (00+11)^*\big((01+10)(00+11)^*(01+10)(00+11)^*\big)^*$, there exists an expression $G' = (01+10)(00+11)^*(01+10)(00+11)^*$ inside the Kleene star. For grammar G', there exists a polynomial grammar $00+11$ inside the Kleene star, indicating that G' is an exponential grammar. Hence, G belongs to the proportional class.

6. Evaluation

Here, our empirical results show that the categorization of RGs is related to the difficulty of RNNs to learn these grammars, and the implementation is publicly available (https://github.com/lazywatch/rnn_theano_tomita.git). We evaluated the three Tomita Grammars 1, 3 and 5. For each grammar, its subclass is shown in Table 3. Following prior work [31], we generated three training sets of binary strings with their lengths ranging from 1 to 14 for these grammars. We also collected three validation sets of strings with different lengths of $[1, 4, 7, \ldots, 25]$, to make sure that the models can be trained to generalize to longer strings. The training process was terminated either when the model achieved a F_1-score that is equal or higher than 0.95 or when a maximum of 5000 epochs were reached. We selected several different recurrent networks to demonstrate how the difficulty of learning generalizes to different models. Specifically, we trained SRN, 2-RNN, GRU and LSTM with data generated on each grammar to a perform a binary classification task. We configured all RNNs to have the same size of hidden layer across all grammars and trained them on each grammar for 10 random trials using a mean squared error loss. In each trial, we randomly initialized the hidden layer of the model.

We followed previous research [20] and used either activation functions—`sigmoid` and `tanh`—to build these RNNs. In addition, for each RNN, we used one-hot encoding to process the input alphabets of 0 s and 1 s. With this configuration, the input layer is designed to contain a single input neuron for each symbol in the alphabet of the target language. Thus, only one input neuron is activated at each time step. Moreover, we followed the approach introduced in previous research and applied the following loss function to all RNNs:

$$L = \frac{1}{2}(y - h_0^T)^2.$$

This loss function can be viewed as selecting a special "response" neuron h_0 and comparing it to the label y, i.e., 1 for acceptance and 0 for rejection. Thus, h_0^T indicates the value of the neuron h_0 at time T after a RNN receives the final input symbol. By using this simple loss function, we attempt to eliminate the potential effect of adding an extra output layer and introducing more weight and bias parameters. In addition, by this design, we ensure that the knowledge learned by a RNN resides in the hidden layer and its transitions. When applying different activation functions, we make sure that h_0 is always normalized between the range of 0 and 1, while other neurons have their values between 0 and 1 for `sigmoid` activation and -1 to 1 for `tanh` activation. During training, we optimized parameters through stochastic gradient descent and employed RMSprop optimization [52].

The results are shown in Figure 5, where the results from all trials fit the shaded area associated with each plot. The x-axis represents the number of epochs during training and the y-axis represents the loss. In Figure 5a–d, we see that Grammars 1 and 3, which have lower entropy values, have learning that converges much more quickly and consistently than that of Grammar 5, which has the highest entropy value. This effect holds for all RNNs evaluated. Note that Grammar 5 defines two sets of strings with equal cardinality when the string length is even. In this case, by flipping any binary digit of a string to its opposite (e.g., flipping a 0 to 1 or vice versa), a valid or invalid string can be converted into a string with the opposite label. This implies that a model must pay equal attention to any string in order to learn the underlying grammar, which makes the learning process more challenging.

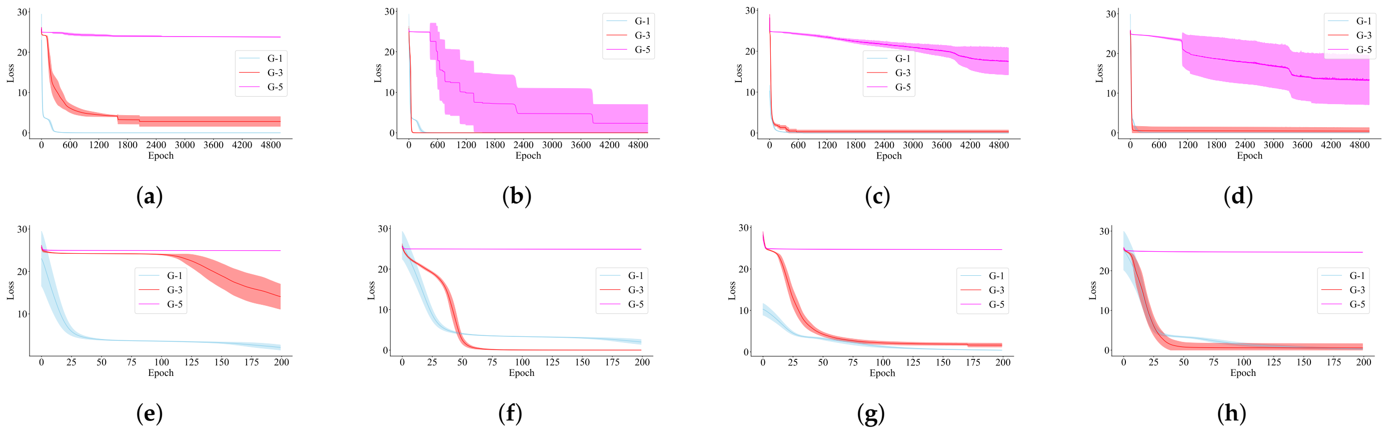


Figure 5. Error loss vs. epochs on Tomita Grammars 1, 3 and 5. (**a**) SRN. (**b**) 2-RNN. (**c**) LSTM. (**d**) GRU. (**e**) SRN-zoom (**f**) 2-RNN-zoom. (**g**) LSTM-zoom. (**h**) GRU-zoom.

To better illustrate the difference of the difficulty of learning Grammars 1 and 3, a zoomed view is provided in Figure 5e–h, for each plot at the top row of Figure 5. While the learning process of all models converges within 100 epochs for both grammars, it is clear that the learning process is slower for Grammar 3. These results agree with both our analysis of the entropy of these grammars and our intuition.

It would be interesting to further investigate the relationship between recurrent models and grammar learning, which is out of the scope for this paper. A promising approach would be to investigate their connection is to more closely represent DFA representations, since both are stateful models. In general, while it is possible to validate the connection between nonlinear RNNs and DFAs empirically, it has been challenging to establish a theoretical connection between nonlinear RNNs and finite state machines. Specifically, second-order RNNs naturally fit into the task of learning any DFA, while some first-order RNNs only represent a portion of DFAs, indicating that the gap (not computationally) between first-/second-order RNNs is not as significant as expected. However, for these gated models (LSTM/GRU), we only observe the differences by experiments.

From the evaluation results, a more general question is to establish an analysis between different types of RNNs and DFAs. To better study this, we discuss the following two questions: Given a RG, does there exist some common substrings misclassified by different RNNs? Given a RNN, are there consistent persistent flaws when learning different RGs? Our results observed on the Tomita grammars can be considered as an initial attempt to answer the above questions. We first find that all RNNs perform perfectly on polynomial RGs (e.g., Grammars 1, 2 and 7), and first-order RNNs perform poorly on proportional RGs (e.g., Grammars 5 and 6). In addition, for first-order RNNs, both their overall classification performance and misclassified strings indicate that these RNNs randomly choose some RGs to learn when learning proportional RGs. For exponential RGs (e.g., Grammar 3), we find that there exist some patterns in strings misclassified by certain first-order and gated RNNs. For example, LSTM tends to ignore "101" appearing at the end of a string and subsequently a has high false-positive errors (and learns some less deterministic transitions). In contrast, the SRN tends to lose count of consecutive 0 s or 1 s. These results indicate that these RNNs learn a smoothed boundary of the manifold that holds these strings. Since these common substrings are more likely to lie in the periphery of the manifold, it suggests the use of up-sampling to compensate for this problem.

7. Conclusions and Future Work

A theoretical analysis and empirical validation for subclasses of regular grammars is presented. Specifically, to measure the complexity of regular grammar, we introduced an entropy metric based on the concentric ring representation, which essentially reflects the difficulty in training RNNs to learn the grammar. Using entropy, we categorized regular grammar into three disjoint subclasses: polynomial class, exponential class and proportional class. In addition, we provided classification theorems for different representations of regular grammar. Given a regular grammar, these theorems use its corresponding properties in the given representation to efficiently determine which subclass a grammar belongs to without calculating the entropy value. Several common representations including deterministic finite automata, regular expression and sets have a corresponding case study which illustrates their applications. This categorization could also be applied to other relevant representations. Finally, we conducted an experiment to demonstrate the influence of grammar learning based on its complexity, which validates the effectiveness of the proposed entropy metric and the theoretical analysis. All RNNs have problems learning certain classes of grammars. It would seem the grammar chosen matters more than the RNN architecture. We believe this work provides a deeper understanding of the internal structure of regular grammar in the context of learning.

Future work could include an extension to other types of grammars, e.g., context-free grammars. The concentric ring representation is independent of grammar type and can be similarly applied to context-free grammars giving similar results to their entropy. However, classification differs dramatically. For instance, when we consider the parentheses grammar, the entropy can be obtained by the central binomial coefficient, which fails to fall in any of the classes proposed in this work. Another perspective is to study grammar learning in terms of recurrent models. Such a connection between DFA and RNNs can provide insights into explainable artificial intelligence and adversarial machine learning.

Author Contributions: Data curation, K.Z. and Q.W.; Formal analysis, K.Z.; Methodology, K.Z.; Software, K.Z. and Q.W.; Supervision, C.L.G.; Writing—original draft, K.Z.; Writing—review & editing, C.L.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

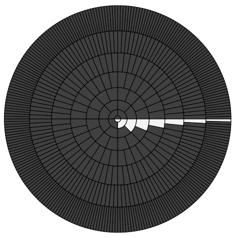
Data Availability Statement: The implementation is publicly available at https://github.com/lazywatch/rnn_theano_tomita.git.

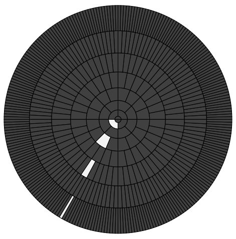
Acknowledgments: We gratefully acknowledge the reviewers for their comments and suggestions. Partial support from the National Science Foundation is gratefully acknowledged.

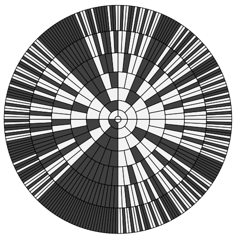
Conflicts of Interest: The authors declare no conflict of interest.

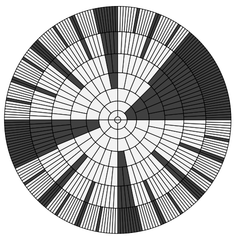
Appendix A. Concentric Ring Representation Representation of Tomita Grammars

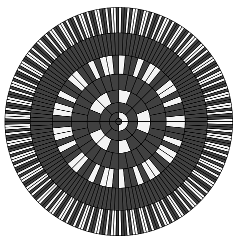
Here, we plot all graphs for Tomita grammars to illustrate their differences.

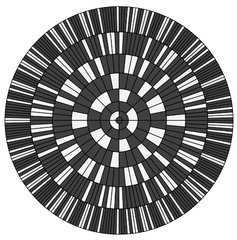

(**a**) Grammar 1.


(**b**) Grammar 2.


(**c**) Grammar 3.


(**d**) Grammar 4.


(**e**) Grammar 5.


(**f**) Grammar 6.

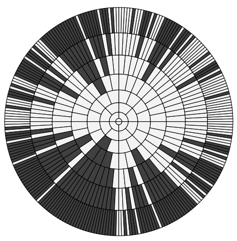

(**g**) Grammar 7.

Figure A1. Concentric ring representation of the distribution of strings of length N ($1 \leq N \leq 8$) for Tomita Grammars. Each concentric ring of a graph has 2^N strings arranged in lexicographic order, starting at $\theta = 0$.

Appendix B. Generalization to the Regular Grammars with an Arbitrary Alphabet

Given a regular grammar G with a I-size alphabet Σ_I and its associated minimal DFA M with n states, let $T_i \in \mathbb{Z}^{n \times n}$ denote the transition matrix of M associated with ith input symbol for $i \in \{1, \dots, I\}$, and $T = \sum_i^I T_i$ is the sum of all transition matrices. We use $k(T)$ to represent the number of diagonal elements equal to I and $\sigma(T)$ to denote the set of modulus of all eigenvalues of T. $|\lambda_2|$ is used to represent the second largest modulus of the eigenvalues of T. Then, its categorization result for the three classes introduced in Theorems 1 and 2 is shown in the following table (without loss of generality here we only derive the result using the number of positive samples, i.e., m_p, defined by this grammar).

Table A1. Generalization to the Regular Grammars with an Arbitrary Alphabet.

Category	Data-Driven Perspective	DFA Perspective	Entropy
[**Po**]	$m_p \sim P(N)$ (polynomial of N)	$k(T) = 1$ and $\sigma(T) = \{1, I\}$	$H(G) = 0$
[**Ex**]	$m_p \sim \beta \cdot b^N$ where $b < I$ and $\beta > 0$	$k(T) = 1$ and $\sigma(T) - \{1, I\} \neq \emptyset$	$H(G) = \log_I \lvert\lambda_2\rvert \in (0, 1)$
[**Pr**]	$m_p \sim \alpha \cdot I^N$, where $\alpha \in (0, 1)$	$k(T) = 0$ or $k(T) = 2$	$H(G) = 1$

References

1. Chowdhary, K. Natural language processing. In *Fundamentals of Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 603–649.
2. Zettlemoyer, L.; Collins, M. Online learning of relaxed CCG grammars for parsing to logical form. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), Prague, Czech Republic, 28–30 June 2007, pp. 678–687.
3. Le Métayer, D. Describing software architecture styles using graph grammars. *IEEE Trans. Softw. Eng.* **1998**, *24*, 521–533. [CrossRef]
4. Wang, Q.; Zhang, K.; Liu, X.; Giles, C.L. Verification of recurrent neural networks through rule extraction. *arXiv* **2019**, arXiv:1811.06029.
5. Parker, A.J.; Yancey, K.B.; Yancey, M.P. Regular language distance and entropy. *arXiv* **2016**, arXiv:1602.07715.

6. Avcu, E.; Shibata, C.; Heinz, J. Subregular Complexity and Deep Learning. *arXiv* **2017**, arXiv:1705.05940.
7. Zhang, K.; Wang, Q.; Liu, X.; Giles, C.L. Shapley Homology: Topological Analysis of Sample Influence for Neural Networks. *Neural Comput.* **2020**, *32*, 1355–1378. [CrossRef]
8. Mikolov, T.; Karafiát, M.; Burget, L.; Černocký, J.; Khudanpur, S. Recurrent neural network based language model. In Proceedings of the Eleventh Annual Conference of the International Speech Communication Association, Chiba, Japan, 26–30 September 2010.
9. Weiss, G.; Goldberg, Y.; Yahav, E. Learning deterministic weighted automata with queries and counterexamples. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, Canada, 3–14 December 2019; pp. 8560–8571.
10. Oliva, C.; Lago-Fernández, L.F. Interpretability of recurrent neural networks trained on regular languages. In Proceedings of the International Work-Conference on Artificial Neural Networks, Gran Canaria, Spain, 12–14 June 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 14–25.
11. Marzouk, R.; de la Higuera, C. Distance and Equivalence between Finite State Machines and Recurrent Neural Networks: Computational results. *arXiv* **2020**, arXiv:2004.00478
12. Marzouk, R. On Computability, Learnability and Extractability of Finite State Machines from Recurrent Neural Networks. *arXiv* **2020**, arXiv:2009.06398.
13. Oliva, C.; Lago-Fernández, L.F. On the interpretation of recurrent neural networks as finite state machines. In Proceedings of the International Conference on Artificial Neural Networks, Munich, Germany, 17–19 September 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 312–323.
14. Huang, J.S. *Lectures on Representation Theory*; World Scientific: Singapore, 1999.
15. Watrous, R.L.; Kuhn, G.M. Induction of finite-state automata using second-order recurrent networks. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 30 November–3 December 1992; pp. 309–317.
16. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [CrossRef]
17. Goudreau, M.W.; Giles, C.L.; Chakradhar, S.T.; Chen, D. First-order versus second-order single-layer recurrent neural networks. *IEEE Trans. Neural Netw.* **1994**, *5*, 511–513. [CrossRef]
18. Hochreiter, S.; Bengio, Y.; Frasconi, P.; Schmidhuber, J. *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies*; Wiley-IEEE Press: Hoboken, NJ, USA, 2001.
19. Cho, K.; Van Merriënboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In Proceedings of the SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014; pp. 103–111.
20. Giles, C.L.; Miller, C.B.; Chen, D.; Chen, H.H.; Sun, G.Z.; Lee, Y.C. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Comput.* **1992**, *4*, 393–405. [CrossRef]
21. Omlin, C.W.; Giles, C.L. Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants. *Neural Comput.* **1996**, *8*, 675–696. [CrossRef] [PubMed]
22. Rabusseau, G.; Li, T.; Precup, D. Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning. AISTATS PMLR. In Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS 2019), Naha, Okinawa, Japan, 16–18 April 2019; Volume 89, pp. 1630–1639.
23. Eyraud, R.; Ayache, S. Distillation of Weighted Automata from Recurrent Neural Networks using a Spectral Approach. *arXiv* **2020**, arXiv:2009.13101.
24. Okudono, T.; Waga, M.; Sekiyama, T.; Hasuo, I. Weighted automata extraction from recurrent neural networks via regression on state spaces. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 Fenruaty 2020; Volume 34, pp. 5306–5314.
25. Sutskever, I.; Martens, J.; Hinton, G.E. Generating Text with Recurrent Neural Networks. In Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, DC, USA, 28 June–2 July 2011; pp. 1017–1024.
26. Wu, Y.; Zhang, S.; Zhang, Y.; Bengio, Y.; Salakhutdinov, R. On Multiplicative Integration with Recurrent Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 5–10 December 2016; pp. 2856–2864.
27. Hopcroft, J.E. *Introduction to Automata Theory, Languages, and Computation*; Pearson Education India: Tamil Nadu, India, 2008.
28. Tomita, M. Dynamic construction of finite-state automata from examples using hill-climbing. In Proceedings of the Fourth Annual Conference of the Cognitive Science Society, Ann Arbor, MI, USA, 4–6 August 1982; pp. 105–108.
29. De la Higuera, C. *Grammatical Inference: Learning Automata and Grammars*; Cambridge University Press: Cambridge, UK, 2010.
30. Li, K.; Príncipe, J.C. The kernel adaptive autoregressive-moving-average algorithm. *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *27*, 334–346. [CrossRef]
31. Weiss, G.; Goldberg, Y.; Yahav, E. Extracting automata from recurrent neural networks using queries and counterexamples. *arXiv* **2017**, arXiv:1711.09576.
32. Hou, B.; Zhou, Z. Learning with Interpretable Structure from RNN. *arXiv* **2018**, arXiv:1810.10708.
33. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
34. Kleene, S. Representation of events in nerve nets and finite automata. *Autom. Stud.* **1951**, *3*, 41.
35. Minsky, M. Neural Nets and the Brain Model Problem. Ph.D. Thesis, Princeton Univ: Princeton, NJ, USA, 1954.

36. Cleeremans, A.; Servan-Schreiber, D.; McClelland, J.L. Finite state automata and simple recurrent networks. *Neural Comput.* **1989**, *1*, 372–381. [CrossRef]
37. Elman, J.L. Distributed representations, simple recurrent networks, and grammatical structure. *Mach. Learn.* **1991**, *7*, 195–225. [CrossRef]
38. Siegelmann, H.T.; Sontag, E.D. On the computational power of neural nets. *J. Comput. Syst. Sci.* **1995**, *50*, 132–150. [CrossRef]
39. Omlin, C.W.; Giles, C.L. Constructing deterministic finite-state automata in recurrent neural networks. *J. ACM* **1996**, *43*, 937–972. [CrossRef]
40. Murdoch, W.J.; Szlam, A. Automatic Rule Extraction from Long Short Term Memory Networks. *arXiv* **2017**, arXiv:1702.02540.
41. Wang, Q.; Zhang, K.; Ororbia, A.G., II; Xing, X.; Liu, X.; Giles, C.L. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Comput.* **2018**, *30*, 2568–2591. [CrossRef] [PubMed]
42. Huang, X.; Kwiatkowska, M.; Wang, S.; Wu, M. Safety verification of deep neural networks. In Proceedings of the International Conference on Computer Aided Verification, Heidelberg, Germany, 24–28 July 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 3–29.
43. Eisner, J.; Gallé, M.; Heinz, J.; Quattoni, A.; Rabusseau, G. Deep Learning and Formal Languages: Building Bridges. In Proceedings of the Workshop on Deep Learning and Formal Languages, Building Bridges, Florence, Italy, 2 August 2019.
44. Lind, D.; Marcus, B.; Douglas, L.; Brian, M. *An Introduction to Symbolic Dynamics and Coding*; Cambridge University Press: Cambridge, UK, 1995.
45. Rogers, J.; Heinz, J.; Fero, M.; Hurst, J.; Lambert, D.; Wibel, S. *Cognitive and Sub-Regular Complexity. Formal Grammar*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 90–108.
46. Pin, J.É. Mathematical foundations of automata theory. In *Lecture Notes LIAFA*; University Paris: Paris, France, 2010; Volume 7.
47. Carrasco, R.C. Accurate computation of the relative entropy between stochastic regular grammars. *RAIRO-Theor. Inform. Appl.* **1997**, *31*, 437–444. [CrossRef]
48. Thollard, F.; Dupont, P.; De La Higuera, C. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In Proceedings of the ICML, Stanford, CA, USA, 29 June–2 July 2000; pp. 975–982.
49. Suresh, A.T.; Roark, B.; Riley, M.; Schogol, V. Distilling weighted finite automata from arbitrary probabilistic models. In Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing, Dresden, Germany, 23–25 September 2019; pp. 87–97.
50. Charalambides, C.A. *Enumerative Combinatorics*; CRC Press: Boca Raton, FL, USA, 2018.
51. Tomita, M. Learning of Construction of Finite Automata from Examples Using Hill-Climbing. RR: Regular Set Recognizer. *Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Dept of Computer Science*; 1982. Available online: https://apps.dtic.mil/sti/citations/ADA120123 (accessed on 14 January 2020).
52. Tieleman, T.; Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* **2012**, *4*, 26–31.