An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks

Qinglong Wang[1], Kaixuan Zhang[2], Alexander G. Ororbia II[2], Xinyu Xing[2], Xue Liu[1], C. Lee Giles[2]

[1]McGill University.

[2]Pennsylvania State University.

Keywords: Deterministic finite automata, second order recurrent neural networks, rule extraction, deep learning, regular grammars.

Abstract

Rule extraction from black-box models is critical in domains that require model validation before implementation, as can be the case in credit scoring and medical diagnosis. Though already a challenging problem in statistical learning in general, the difficulty is even greater when highly non-linear, recursive models, such as recurrent neural networks (RNNs), are fit to data. Here, we study the extraction of rules from second order recurrent neural networks trained to recognize the Tomita grammars. We show that production rules can be stably extracted from trained RNNs and that in certain cases the rules outperform the trained RNNs.

1 Introduction

Recurrent neural networks (RNNs) have been increasingly adopted for a variety of tasks involving time-varying data, e.g. sentiment analysis, machine translation, image captioning, etc. Despite the impressive performance on these tasks, RNNs are also well-known to be “black-box” models, which makes explaining/interpreting the knowledge acquired by these models difficult or near-impossible. This black-box nature is largely due to the fact that RNNs, much as any neural architecture, e.g. convolutional neural networks, although designed to capture structural information from the data (Du et al., 2017), store learned knowledge in their weights, which is difficult to inspect, analyze, and verify (Omlin & Giles, 2000).

Given RNN’s rising popularity in processing time-varying data, we investigate whether and how we might extract knowledge in symbolic form from RNN models that have been trained on symbolic data, in this case a collection of regular grammars. Surprisingly, this is an old problem treated by Minsky in the chapter titled ”Neural Networks. Automata Made up of Parts” in his text ”Computation, Finite and Infinite Machines” (Minsky, 1967). If the information processing procedure of the RNN can be treated as representing knowledge in symbolic form, where a set of rules that govern transitions between symbolic representations are learned, then we can begin to view the RNN as an automated reasoning process that can be easier to understand. Indeed, prior work (Borges et al., 2011) has proposed to extract symbolic knowledge from a nonlinear autoregressive models with exogenous (NARX) recurrent model (Lin et al., 1996). For sentiment analysis tasks, recent work (Murdoch & Szlam, 2017) has demonstrated that an RNN is capable of identifying consistently important patterns of words. These words can be viewed as symbolic knowledge and the patterns of these words represents the rules for determining the sentiment. In other work (Dhingra et al., 2017), information about long-term dependencies are also represented in the form of symbolic knowledge to improve the ability of RNNs to handle long-term text data. Also, prior work (Giles et al., 1992; Watrous & Kuhn, 1992; Omlin & Giles, 1996c; Casey, 1996;

Jacobsson, 2005) has shown that it is possible to extract deterministic finite automata (DFA) from RNN models trained to perform grammatical inference and that grammatical rules can be stably encoded and represented in second order RNNs (Omlin & Giles, 1996a,b). In these studies, the vector space of an RNN's hidden layer is first partitioned into finite parts, each treated as the states of a certain DFA. Then, transitions rules between these states are extracted. This paper follows the paradigm of DFA extraction laid out in previous research.

While it has been shown that it is possible to extract DFA from RNNs, it has been argued (Kolen, 1994) that DFA extraction is sensitive to the initial conditions of the hidden layer of RNN. In other words, by viewing an RNN as a nonlinear dynamical system, the value of its hidden layer may exhibit exponential divergence for nearby initial state vectors. As a result, any attempts at partitioning the hidden space may result in forcing the extracted state to split into multiple trajectories independent of the future input sequence. This results in an extracted rule that appears as a nondeterministic state transition, even though underlying dynamical system is completely deterministic (Kolen, 1994).

In this paper, we greatly expand upon previous work in rule extraction from second-order RNNs (Giles et al., 1992) by studying DFA extraction through comprehensive experiments. The main questions that we hope to ask are:

1. What conditions will affect DFA extraction and how sensitive is DFA extraction with respect to these conditions?

2. How well will the extracted DFA perform in comparison with the RNN trained models from which they are extracted?

With respect to the first question, we aim at uncovering the relationship between different conditions, for instance, the influence of the initial condition of the RNN's hidden layer and the configuration of adopted clustering algorithm on DFA extraction. Specially, through our empirical study, we address the concerns of (Kolen, 1994) by showing that DFA extraction is very insensitive to the initial conditions of the hidden layer.

Moreover, in answering the second question, we find that in most cases the extracted DFA can recognize a set of strings generated by a certain regular grammar as accurately as the trained RNN models from which the DFA were extracted. Interestingly enough, in certain cases, we observe that extracted DFA even outperform their source RNNs in term of recognition accuracy when processing long sequences. This result is surprising given the difficulty in training RNNs on long sequences, largely due to the vanishing gradient problem (Pascanu et al., 2013), of which a great deal of research has been dedicated to solving (Hochreiter & Schmidhuber, 1997; Cho et al., 2014; Weston et al., 2014; Sukhbaatar et al., 2015; Dhingra et al., 2017). Extracting rules from RNNs also sheds light on an alternative to improve the processing of long pattern sequences.

Here, our emphasis is on examining the consistency of DFA extraction. More specifically, we first train and test RNN models on data sets generated by the seven Tomita grammars (Tomita, 1982a). The RNN models we use have a second-order architecture (Giles et al., 1992). Then we collect the values of hidden layer units of RNN models obtained during the testing phase, and cluster these values. Here we use k-means due to its simplicity and efficiency. We believe other clustering methods could provide similar results. These clustered states and the symbolic inputs are used to form the initial DFA, which may contain equivalent states. Finally, we use a minimization algorithm to minimize the number of states and finalize the minimal DFA.

In summary, this work makes the following contributions.

- We conduct a careful experimental study of the factors that influence DFA extraction. Our results show that, despite these factors, DFA can be stably extracted from second-order RNNs. In particular, we find strong evidence that, by adopting a simple clustering method, DFA can be reliably extracted even when the target RNN is only trained using short sequences.

- We explore the impact of network capacity and training time on the RNN's ability to handle long sequences and find that these factors play key roles. With respect to DFA extraction, however, these factors exhibit only a limited impact. This shows that extracting DFA requires less effort compared to the training of a powerful RNN.

- We investigate a realistic case where “incorrect” DFA are extracted from low capacity second-order RNNs, and demonstrate that, in some cases, these DFA can still outperform the source RNNs when processing long sequences. This sheds light on a possible path to improving an RNN’s ability in handling long sequences–exploiting the DFA’s natural ability to handle infinitely long sequences (which is a challenge for any RNN).

2 Background

Recurrent neural networks (RNN) process sequential data by encoding information into the continuous hidden space in an implicit, holistic manner (Elman, 1990). In order to extract rules from this continuous hidden space, it is commonly assumed that the continuous space is approximated by a finite set of states (Jacobsson, 2005). The rule is then referred to as the transitions among the discrete states. A common choice for representation of the extracted rules is a DFA. In the following, we first provide a brief introduction of DFA, followed by an introduction to the target grammars studied. Finally, we present a particular type of RNN – a second-order RNN, which is mainly used in this work.

2.1 Deterministic Finite Automata

A finite state machine M recognizes and generates certain grammar G, which can be described by a five-tuple $\{A, S, s_0, F, P\}$. Here, A is the input alphabet (a finite, non-empty set of symbols), S is a finite, non-empty set of states. $s_0 \in S$ and $F \in S$ represents the initial state (an element of S) and the set of final states (a subset of S, F can be empty). P denotes a set of production rules (transition function $P : S \times A \to S$). Every grammar G also recognizes and generates a corresponding language $L(G)$, a set of strings of the symbols from alphabet A. The simplest automata and its associated grammar are DFA and regular grammars, according to the Chomsky hierarchy of phrase structured grammars (Chomsky, 1956). It is important to realize DFA actually covers a wide range of languages, that is, all languages whose string length and alphabet size are bounded can be recognized and generated by finite state automata (Giles et al., 1992). Also, when replacing the deterministic transition with stochastic transition, a DFA can be converted as a probabilistic automata or hidden Markov model, which enables grammatical inference as learning on graphical models (Du et al., 2016). We refer the reader to a more detailed introduction of regular languages and finite state machines in (Hopcroft et al., 2013; Carroll & Long, 1989) and use their notation.

2.2 Tomita Grammars

We select a set of seven relatively simple grammars, which are originally suggested by Tomita (Tomita, 1982a) and widely studied (Pollack, 1991; Omlin & Giles, 1996c; Watrous & Kuhn, 1992) and use them for an empirical study for extracting rules from RNN. We hypothesize (and note from the work of others) that these simple regular grammars should be learnable. More specifically, the DFA associated with these grammars have between three and six states. These grammars all have $A = \{0, 1\}$, and generate an infinite language over $\{0, 1\}^*$. Here we denote a finite set of strings I from regular language $L(G)$. Positive examples of the input strings are denoted as I_+ and negative examples as I_-. We provide a description of positive examples accepted by all seven grammars in Table 1.

The associated DFA for these grammars is shown in the first column in Figure 5. Some of these DFA contain a so-called “garbage state”, that is, a non-final state in which all transition paths lead back to itself. In order to correctly learn this state, RNN must not only learn with positive strings I_+ generated by the grammar, but also negative strings I_- that are rejected by this grammar.

Despite the fact that the Tomita grammars are relatively simple, we select these

Table 1: Description of seven Tomita grammars

G	Description
1	1*
2	(10)*
3	an odd number of consecutive 1s is always followed by an even number of consecutive 0s
4	any string not containing “000” as a substring
5	even number of 0s and even number of 1s (Giles et al., 1990)
6	the difference between the number of 0s and the number of 1s is a multiple of 3
7	0*1*0*1*

grammars because they actually cover regular languages that have different complexity and difficulty (Wang et al., 2018). They also appear to be a standard for much work on learning grammars. For example, grammars 1,2 and 7 represent the class of regular languages that define a string set that has extremely unbalanced positive and negative strings. This implies that the averaged difference between positive strings and negative strings can be very large. This could represent real-world cases where positive samples are significantly outnumbered by negative ones. In contrast, grammars 5 and 6 define the class of regular languages that have equal or a relatively balanced number of positive and negative strings. This implies that the difference between positive and negative strings in these grammars is much smaller than the case of grammars 1,2 and 7. Finally, grammars 3 and 4 represent the class of regular languages for which the difference between positive and negative strings is somewhere between the above two cases. With either case discussed, the source RNNs are forced to recognize the various levels of difference between positive and negative samples. In addition, it is also important to note that we have ground truth DFAs for Tomita grammars. This enables this study to determine the impact of different factors on the success rate of extracting correct DFAs (introduced in Section 4), since they can be compared to the ground truth DFAs. With more complex/or real-world datasets, this may not be case. For those datasets, uncertainties will be introduced into the evaluation (e.g. what is the ground truth DFA or if there even exists ground truth DFAs that define the data?). This uncertainty can affect any conclusion of whether a DFA extraction can be stably performed.

2.3 Second-order Recurrent Neural Networks

Here, we use an RNN constructed with second-order interactions between hidden states and input symbols. More specifically, this second-order RNN has a hidden layer H containing N recurrent hidden neurons h_i, and L input neurons i_l. The second-order interaction is represented as $w_{ijk}h_j^t i_k^t$, where w_{ijk} is a $N \times N \times L$ real-valued matrix, which modifies a product of the hidden h_j and input i_k neurons. t denotes the tth discrete time slot. This quadratic form directly represents the state transition diagrams of a state process – $\{$input, state$\} \Rightarrow \{$next state$\}$. More formally, the state transition is defined by following equation:

$$H_i^{t+1} = g(\sum_{j,k} W_{ijk} H_j^t I_k^t) \tag{1}$$

where g is a sigmoid discriminant function. Each input string is encoded by one-hot-encoding, and the neural network is constructed with one input neuron for each character in the alphabet of the relevant language. By using one-hot-encoding, we ensure that only one input neuron is activated per discrete time step t. Note that, when building a second-order RNN, as long as L is small compared to N, the complexity of the network only grows as $O(N^2)$. Such RNNs have been proved to stably encode finite state machines (Omlin & Giles, 1996a,b) and thus can represent in theory all regular grammars.

To train above second-order RNN, we use the following loss function C following (Giles et al., 1992):

$$C = \frac{1}{2}(y - H_0^T)^2 \tag{2}$$

where C is defined by selecting a special "response" neuron h_0, which is compared to the target label y. For positive strings, $y = 1.0$ and $y = 0.0$ for negative strings.

h_0^T indicates the value of h_0 at time T after seeing the final input symbol. We adopt *RMSprop* (Tieleman & Hinton, 2012) as the training algorithm.

3 DFA Extraction

We introduce our approach to DFA extraction, which largely builds on the research conducted in the 1990's (please see many of the citations in the bibliography) but note that there has been recent work (Li & Príncipe, 2016). We start by briefly introducing the main ideas behind DFA extraction as well as existing research. We will then examine and identity key factors that affect the quality of each step of the extraction process.

3.1 The DFA Extraction Paradigm

Many methods have been developed to extract knowledge in the form of rule from trained RNNs (Giles et al., 1991, 1992; Omlin & Giles, 1996c; Zeng et al., 1993; Frasconi et al., 1996; Gori et al., 1998). Most of this work can be viewed as roughly following one general DFA extraction process:

1. Collect the hidden activations of RNN when processing every string at every time step. Cluster these hidden activations into different states.

2. Use the clustered states and the alphabet-labeled arcs that connect these states to construct a transition diagram.

3. Reduce the diagram to a minimal representation of state transitions.

Previous research effort has largely focused on improving the first two steps. This is largely due to the fact that, for the third step, there already exists a well established minimization algorithm (Hopcroft et al., 2013) for obtaining the minimal representation of DFA.

For the first step, an equipartition-based approach (Giles et al., 1992) was proposed to cluster the hidden space by quantizing the value of a hidden unit to a specified number of bins. For example, if we apply a binary quantization [1] to the vector $\{0.6, 0.4, 0.2\}$, we would obtain the encoding $\{1, 0, 0\}$. One drawback to this form of quantization is that as the number of hidden units increases, the number of clusters grows exponentially. This computational complexity issue is alleviated if one uses clustering methods that are less sensitive to the dimensionality of data samples, e.g., k-means (Zeng et al., 1993; Frasconi et al., 1996; Gori et al., 1998), hierarchical clustering (Sanfeliu & Alquezar, 1994), and self-organizing maps (Tiňo & Šajda, 1995).

In order to construct state transitions for the second step, either breadth-first search (BFS) approaches (Giles et al., 1992) or sampling-based approaches (Tiňo & Šajda, 1995) are utilized. BFS approach can construct a transition table relatively consistent but incur high computation cost especially when the size of alphabet increases exponentially. Compared with BFS approach, sampling approach is computationally efficient. However, it introduces inconsistency to the construction of a transition table. For a more detailed exposition of these two classes of methods, we refer the readers to this survey (Jacobsson, 2005).

3.2 Factors That Affect DFA Extraction

The efficacy of the different methods used for the first two steps of the process described above rely on the following hypothesis: The state space of a well-trained RNN should already be fairly well-separated, with distinct regions or clusters that represent corresponding states in some DFA. This hypothesis, if true, would greatly ease the process of DFA-extraction. In particular, less effort would be required in the first two steps of DFA extraction if the underlying RNN was constructed to have a well-separated state-space.

With this in mind, we specify the following key factors that affect DFA extraction that also affect representational ability of an RNN.

- Model capacity. A RNN with greater capacity (larger size of hidden layer) is

[1] Using a threshold value of 0.5, any value greater than 0.5 is assigned to the bin “1”, whereas other values less than or equal to this threshold are assigned to “0”.

more likely to better represent a DFA.

- Training time. A sufficient number of iterations are required in order to ensure convergence (to some local optima).

- Initial conditions of the hidden state. As argued previously (Kolen, 1994), the initial conditions may have significant impact on DFA extraction. In this work, we explore this impact by training several RNN models with random initial hidden activations on all grammars, then examining the extracted DFA from all trained RNN models.

- Choice of state-clustering method. The choice of clustering algorithm is very important, including its hyper-parameter configuration. For example, if k-means or a Gaussian mixture model is adopted, a critical hyper-parameter is the predefined number of clusters.

One could argue that other factors, such as choice of a parameter update rule (e.g., ADAM, RMSProp, etc.) and learning rate, may also influence how well an RNN learns about certain grammar. However, in our experiments, we observe that these latter conditions actually have little and nearly no influence on the final results. Thus, we focus on the factors described in the list above.

3.3 The DFA Extraction Process

Here, we use an approach similar to Zeng et al. (1993) to extract DFA from second-order RNNs. To be more specific, we first train second-order RNNs to classify strings generated by each of the seven Tomita grammars (Tomita, 1982b). A desirable outcome of the hypothesis described in the previous section is that, when the hidden space is well-separated, many well-established clustering methods should generate similar results. This allows us to choose our clustering approach based on computational complexity. As a result, we adopt the k-means clustering approach due to its simplicity and efficiency. However, as will be discussed in the following, we must now concern ourselves with choosing an appropriate value of K.

After clustering the hidden space, we follow the approach taken in Schellhammer et al. (1998) to construct the transition diagram. Specifically, we construct the diagram by counting the number of transitions that have occurred between a state and its subsequent states (given a certain input). For example, given a state S_k and input symbol i, we calculate the number of transitions to all states $\{S\}$ from S_k, including any self-loops. After obtaining the transition counts, we keep only the most frequent transitions between $\{S\}$ and $\{S+1\}$ given input i and discard the other less frequent ones in the transition diagram.

It is important to note that K should not be set too small. In an extreme case, when the value of K is set to be even smaller than the minimal number of states of the ground truth DFA, the extraction never provides the correct DFA. Additionally, when K is small, the hidden activations that should have formed different clusters (which represent different states) may be forced to be included in a single cluster, hence generating poor clustering. We illustrate this effect by demonstrating in Figure 1 the clustering obtained by selecting different K's. More specifically, we evaluate the clustering using a silhouette coefficient to measure how well the resulting clusters are separated. As shown in Figure 1, when K is smaller than 6, the clustering is much less desirable and varies significantly than when K is larger. This poorly clustered hidden space will more likely cause inconsistent transitions between states given the same input. For example, assuming there are two cluster S_1 and S_2, given the same input symbol i, they transit to S_3 and S_4 respectively. When K is small, it is possible that S_1 and S_2 are merged as one cluster $\hat{S}_1$. As a result, $\hat{S}_1$ will inconsistently visit S_3 and S_4 with the same input i. This falsely indicates that the transition learned is more likely to be non-deterministic, while the real case is that the RNN generates S_{t+1} based on S_t and i deterministically. This effect can be mitigated when K is increased beyond a certain value. However, this does not indicate that K can be set arbitrarily large. Larger K only brings limited improvement in the clustering results, while imposing more computation on both the

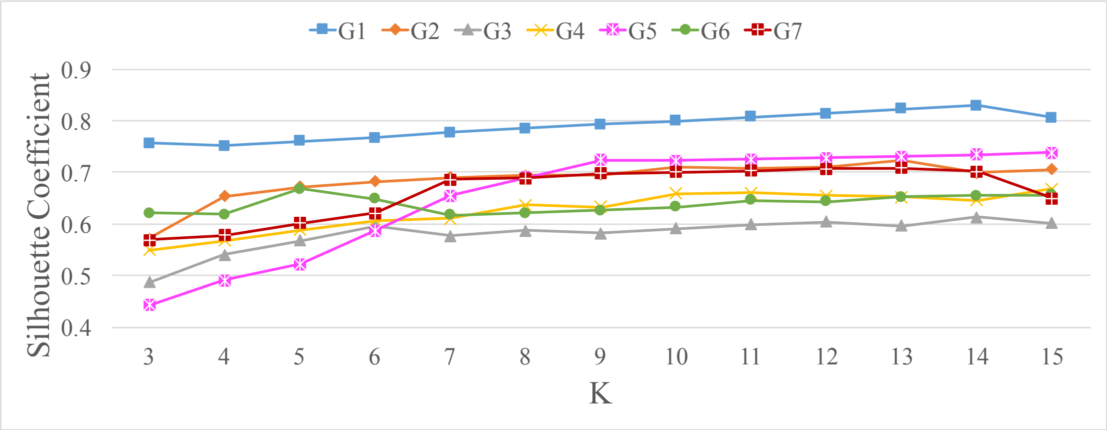


Figure 1: Influence of K on clustering results for all grammars

clustering algorithm and the minimization algorithm which is introduced next.

With the constructed transition diagram, we have extracted a DFA that might contain many redundant states. Using the previously described minimization algorithm (Hopcroft et al., 2013), we can then reduce the derived DFA to its minimal representation. Note that this minimization algorithm does not change the performance of the DFA; the unminimized DFA has the same time complexity as the minimized DFA. Note that the DFA extraction method introduced above may be applied to any RNN, regardless of order or manner in which its hidden layers are calculated.

4 Experiments

In this section, we empirically study the process of DFA extraction through comprehensive experiments.

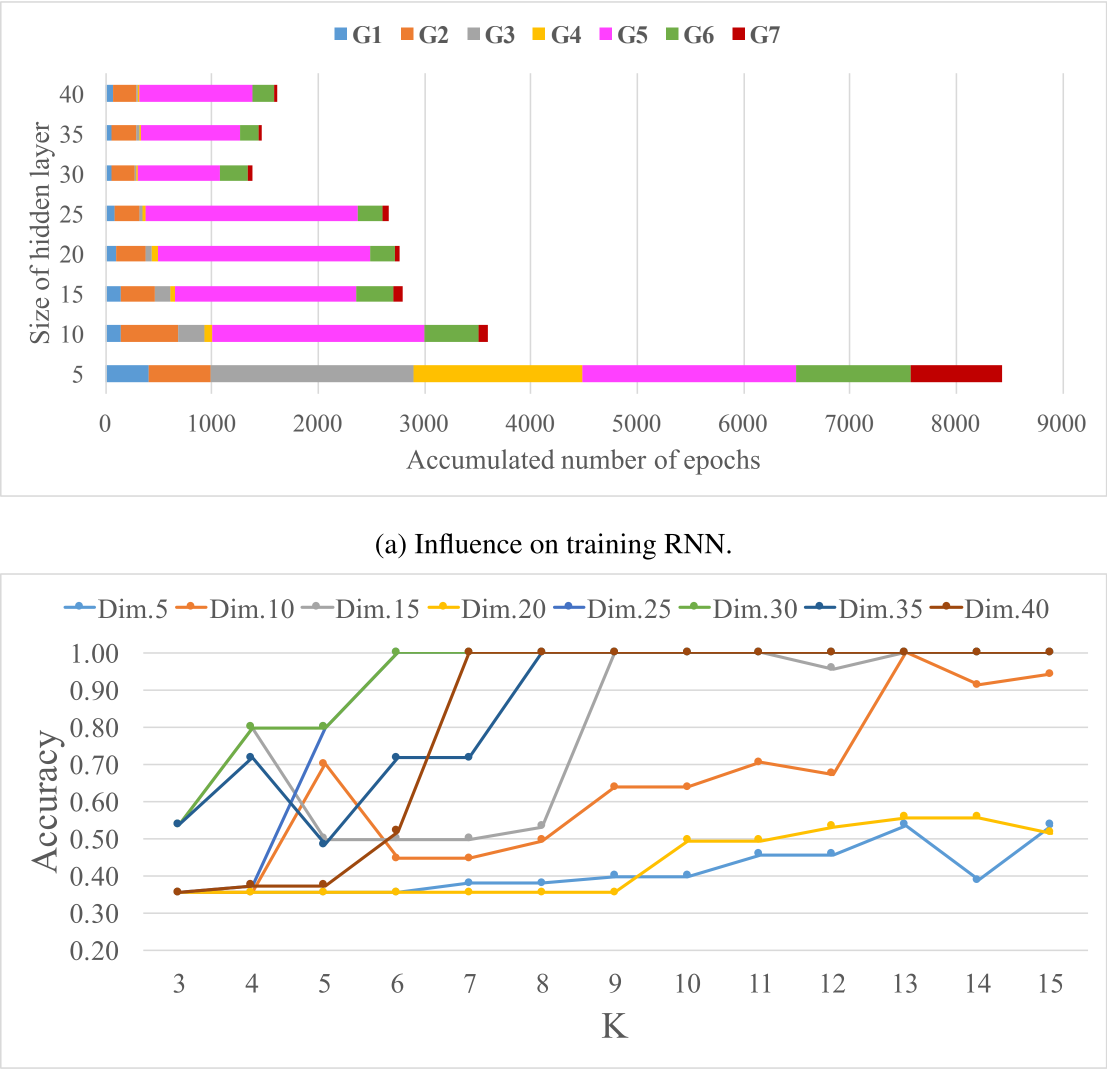
4.1 Description of Data

To train and test the RNN models, we followed the approach introduced in Giles et al. (1992) and generated string sets. To be specific, we drew strings from an oracle generating random 0 and 1 strings and the grammar specified in Table 1. The end of each string is set to symbol 2, which represents the "stop" symbol (or end-token as in language modeling). For the strings drawn from a grammar, we took them as positive samples while those from that random oracle as negative samples. Note that we verified each string from the random oracle and ensured they are not in the string set represented by that corresponding grammar before treating them as negative samples. It should be noticed that for each grammar in our experiments represents one set of strings with unbounded size. As such we restricted the length of the strings used with a upper bound equal to 15 for all grammars. In addition, we also specify a lower bound on the string lengths to avoid training RNNs with empty strings. In order to use as many strings as possible to build the datasets, the lower bound should be set to be sufficiently small. In our experiments, we set the lower bound equal to 3 for all the grammars. We split the strings generated within the specified range of length for each grammar to build the training set D_{train} and testing set D_{test}, then trained and tested the RNNs accordingly.

In order to further the trained RNNs and extracted DFA on longer strings, we build another testing set $D_{test(200)}$ comprised of strings of length 200 for all grammars. Note that the complete set of strings with length 200 numbers around 10^{60}. A test set of this size is too expensive and not even necessary for evaluating RNNs or DFA. Therefore, we construct the testing set by randomly sampling 100,000 strings for all grammars. In addition, to preserve the actual balance of positive to negative samples, we sample such that we preserve their original proportions as measured from the original, complete set of length 200 strings. For example, for grammar 5, we sample positive and negative strings with the same ratio of 0.5.

4.2 The Influence of Model Capacity

In following experiment, we first measure the influence of model capacity, i.e. the size N of hidden layer of RNN models, on learning the target DFA. Specifically, we measure the training time needed for RNNs with different hidden layer sizes to reach perfect accuracy on the testing set D_{test} for all grammars. It is clear from Figure 2a that



(a) Influence on training RNN.

(b) Influence on extracting DFA.

Figure 2: The influence of model capacity on DFA extraction for grammar 3.

it takes less training for an RNN with a larger capacity N to converge. This is what we would expect; an RNN with larger capacity, in general, can better fit the data.

Next, we evaluate how stably correct DFA can be extracted from the trained RNN models. Here we argue that DFA extraction should be more stable from a RNN model for which the hidden state space is well separated. Intuitively, a well-separated space means that with a well trained second-order RNN, hidden activations obtained after processing a set of strings have already aggregated into different regions that are separated from each other in the hidden node space. In this case, it would be more flexible to select a different K to cluster this space. Assuming the ground truth value of K is M, as when K is larger than M, K-means can already identify M large clusters that contain the majority of the hidden activations. For the other $M - K$ clusters, they only identify outliers. This is also verified in Figure 1. Specifically, when K is sufficiently large, the silhouette coefficient changes slightly as K increases. This is because the small clusters formed by outliers only contribute trivially to the calculated silhouette coefficient. These small clusters will later be recognized as a redundant state and are eliminated by the minimization algorithm. As such, we believe that a more flexible choice of K indicates that the hidden space has already been well separated. To examine this flexibility, we vary K within a certain range to check the accuracy of the extracted DFAs. When more correct DFAs can be extracted from a model, we then determine the choice of K as being more flexible, thus indicating that this model has its hidden space better separated.

From the above discussion for models with a different number of hidden neurons, we compare the classification accuracy on D_{test} of the extracted DFA when increasing K from 6 to 15 on grammar 3. Similar results for the other grammars are provided in the Appendix. As shown in Figure 2b, models with the number of hidden neurons larger than 10 allow more flexible choices for K. For instance, when $N > 20$, the correct DFA can be reliably extracted in most cases of K from 3 to 16. On the contrary, for models with less hidden neurons, the range of K that produces correct DFAs is more limited. For instance, when $N = 5$, the extraction fails for all K within the same range. In addition, when N is larger than 25, successful extraction is only observed when K is larger than 8. These results also indicate that DFA extraction is more likely to succeed when K is set to larger values. This observation is consistent with the results reported in Zeng et al. (1993).

The above experimental results indicate that RNNs with larger capacity are more likely to automatically form a reasonably well-separated state space. As a result, the extraction of DFA is less sensitive to the hidden state clustering step of the process.

Table 2: Influence of training time on DFA extraction and RNN performance.

Grammar	Classification errors reached under different training epochs.							
G3	Epoch	10	20	30	40	50	60	70
	RNN(D_{test})	0.99	0.99	0.99	1.0	1.0	1.0	1.0
	RNN($D_{test(200)}$)	9.1e-2	0.55	0.51	0.81	1.0	0.96	0.86
	DFA	1.0	1.0	1.0	1.0	1.0	1.0	1.0
G4	Epoch	10	20	30	40	50	60	70
	RNN(D_{test})	0.98	0.99	0.99	1.0	1.0	1.0	1.0
	RNN($D_{test(200)}$)	3.3e-3	3.6e-3	4.4e-3	9.7e-3	4.1e-3	8.9e-3	7.8e-3
	DFA	1.4e-3	1.2e-3	2.4e-3	1.0	1.0	1.0	1.0
G5	Epoch	600	650	700	750	800	850	900
	RNN(D_{test})	0.63	0.29	0.46	0.44	1.0	1.0	1.0
	RNN($D_{test(200)}$)	0.63	0.3	0.46	0.45	1.0	1.0	1.0
	DFA	0.67	0.67	0.67	0.67	1.0	1.0	1.0

4.3 The Influence of Training Time

In this part, we evaluate the classification performance of both trained RNNs and extracted DFA when processing longer strings. More specifically, we measure the classification errors made by both RNNs and DFA on the test set $D_{test(200)}$, as shown in Table 2. For example, with respect to grammar 3, we train seven RNNs with different training epochs (increasing from 10 to 70). Seven DFA are then extracted, the testing performance of each as a function of epoch is displayed in Table 2. Due to the space restriction, here we only show the results obtained for grammars 3,4 and 5. The results for other grammars are provided in the Appendix.

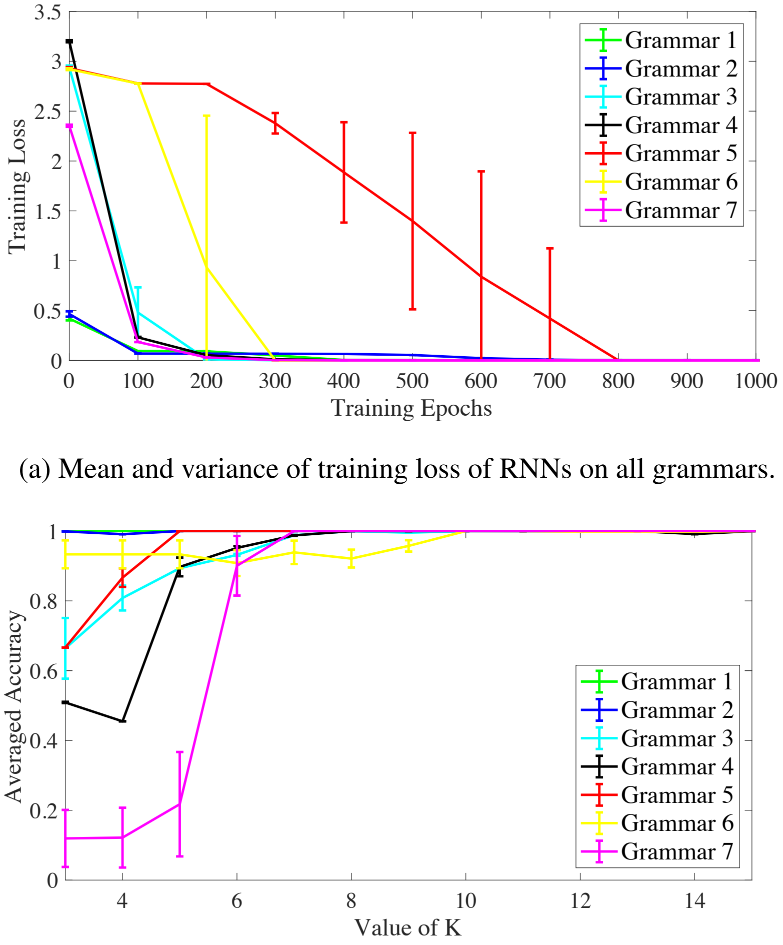
As expected, as the training time increases, RNNs tend to make more accurate classification. In particular, for grammars 3, 4 and 5, the trained RNNs reach 100% accuracy on D_{test}. We observe that the correct DFA can sometimes be extracted even when the RNN has not yet fully reached 100% accuracy (10th to 30th epoch for grammar 3). This indicates that the hidden state space learned in the early stages of training (before the

RNN is fully trained) can still be sufficient for a clustering method to recognize each individual state. This observation implies that less effort is needed to extract the correct DFA from an "imperfect" RNN than in training a "perfect" RNN.

Two other interesting observations can be made with respect to grammar 5. First, it takes much longer to both train an RNN to achieve perfect classification performance and extract the correct DFA from it. Second, the correct DFA can only be successfully extracted when the source RNN quits making any mistakes on the test sets. The difficulty behind training on grammar 5 might be explained through examination of the "differences" between the positive and negative strings generated by the grammar. More specifically, by flipping 1 bit of a string from 0 to 1, or vice versa, any positive (negative) string can be converted to a negative (positive) string. In order to learn the small differences, an RNN needs significantly more training time. The second observation may be explained by noting that, before reaching 800 epochs, RNNs make a nearly constant number of errors. This clearly indicates that the RNN is stuck at a certain local minima (also verified in Figure 3a). While the training of RNN is trapped in this minima, the state space does not start to form the correct partition. However, after 800 epochs, the model escapes this minima and finally converges to a better one, resulting in a state space that is separated correctly.

4.4 The Influence of Initial States & Clustering Configuration

In the following experiments, we examine if a DFA can be stably extracted under random initial conditions. Specifically, for each grammar, we randomly assign an initial value to the hidden activations, i.e. $H^0_{0:N}$ at t_0 time step, within the interval of $[0.0, 1.0]$. We repeat this random initialization ten times (training ten different RNNs) for each grammar. Furthermore, we vary the value of K for the k-means clustering algorithm, measuring the classification performance of each extracted DFA, and counting the number of times the correct DFA is extracted (only DFA achieving 100$\%$ accuracy are regarded as correct). Through this procedure, we hope to uncover the relationship



(a) Mean and variance of training loss of RNNs on all grammars.

(b) Mean and variance of testing accuracy of extracted DFA with varying K on all grammars.

Figure 3: Influence of random initialized hidden activations and clustering configuration on training RNN and extracting DFA.

between the initial condition of the RNN's hidden layer as well as the clustering algorithm's meta-parameter K and DFA extraction.

As previously discussed, training an RNN properly is critical for successful DFA extraction. In Figure 3a, we show the mean and variance of the training loss obtained when training each RNN with 10 times of random initialization of hidden activation for all grammars. It is clear from Figure 3a that, except for grammar 5 and 6, RNNs trained on other grammars rapidly converge. For grammar 5 and 6, RNNs need much more training time while having much larger variance of training loss. Recall above discussion in Section 4.3, this is a clearer indication that the training of these RNNs is trapped to different local optima with different initial activation. However, when given sufficient training, all RNNs trained on all grammars converge on the training set and reach 100% accuracy on D_{test}. In addition, once these RNNs converge to small training loss, the variance reduce to almost 0. This indicates that, with sufficient training and reasonable capacity, RNN's training is relatively insensitive to the hidden layer's initialization.

Given the RNNs trained as described above, we then vary K as we extract DFA from these models. Similarly, we report the mean and variance of the classification accuracy obtained on D_{test} from all extracted DFA in Figure 3b. To be more specific, for each grammar, under each random initialization of the model's hidden layer, we run the extraction process 13 times, varying K in the range from 3 to 15. In total, we conduct 130 rounds of DFA extraction from the ten trained RNNs for each grammar.

As shown in Figure 3b, when K is set to small values (below 8), except for grammar 1 and 2, the extracted DFA on other grammars have not only poor classification accuracy, but also relatively large variance. In this case, it is difficult to determine whether random initialization of hidden activation or K have stronger impact on the extraction. When K is set to a sufficiently large value, however, the variance is significantly reduced while the classification accuracy is greatly improved. This indicates that a sufficiently large K can offset the impact of initial states.

Besides showing the classification performance obtained by the extracted DFA, we further measure the success rate of extraction in Figure 4 under different K. More specifically, the success rate of extraction is the percentage of DFAs with 100.0% accuracy among all DFAs extracted for each grammar under different settings of K and random initializations. Among all 130 rounds of extraction on each grammar, we observe that the correct DFA is successfully extracted with highest success rate of 100.0% (on grammar 1), lowest success rate of 50.0% (on grammar 3) and averaged success rate of 75.0% among all grammars. The reason for the worse extraction results obtained on grammar 3 can be explained by visualizing the extracted DFA in Figure 5.

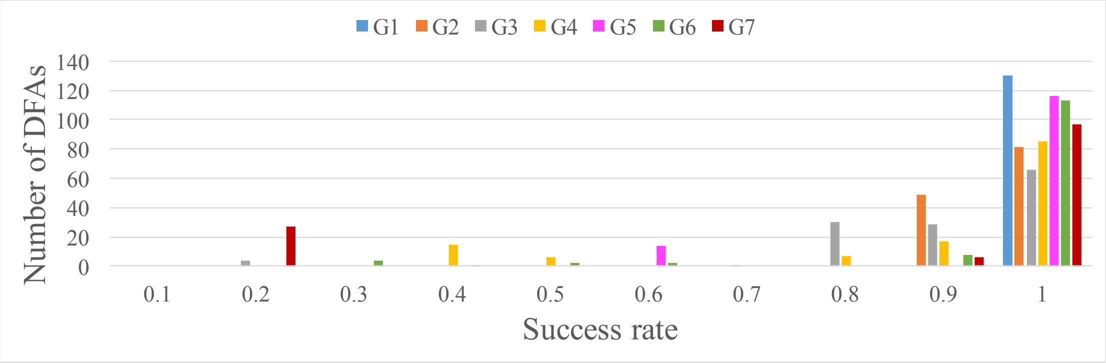


Figure 4: Histograms of the classification performance of extracted DFA on all grammars. The vertical axis is displayed in log-scale.

In Figure 5, we see that all extracted DFA can correctly recognize their associated positive strings, except for ones have length smaller than the minimal length we set. Recall that for all grammars, we generate strings while constraining the minimal string length. The visualization indicates that the extracted DFA not only accurately represent the target grammar that generated the string samples, but also obey the constraint on length. In order for the extracted DFA to satisfy the minimal length constraint, extra states are required, as shown in right panel of Figure 5. Especially, for grammar 3, 4 and 7, the correct DFA contain 5, 4 and 5 states, while the corresponding extracted DFA have 6, 5 and 7 states, respectively. Recall that in above experiments, the minimal value of K is set to 3 consistently for all grammars. As a result, this setting of K causes many failures of extraction for these grammars. As shown in Figure 3b, when K is below 8, the averaged classification accuracy of the extracted DFA are relatively lower in comparison with DFA extracted from other grammars.

4.5 Comparison between Low-Capacity RNNs and Extracted DFA

As discussed in Section 4.2, RNNs with larger capacity can learn a DFA better. In practice, it is usually not possible to know the appropriate capacity when constructing an RNN. As a result, it is possible that a smaller RNN can be well-trained on short sequences, but will generalize poorly when confronted with long sequences. Above ex-

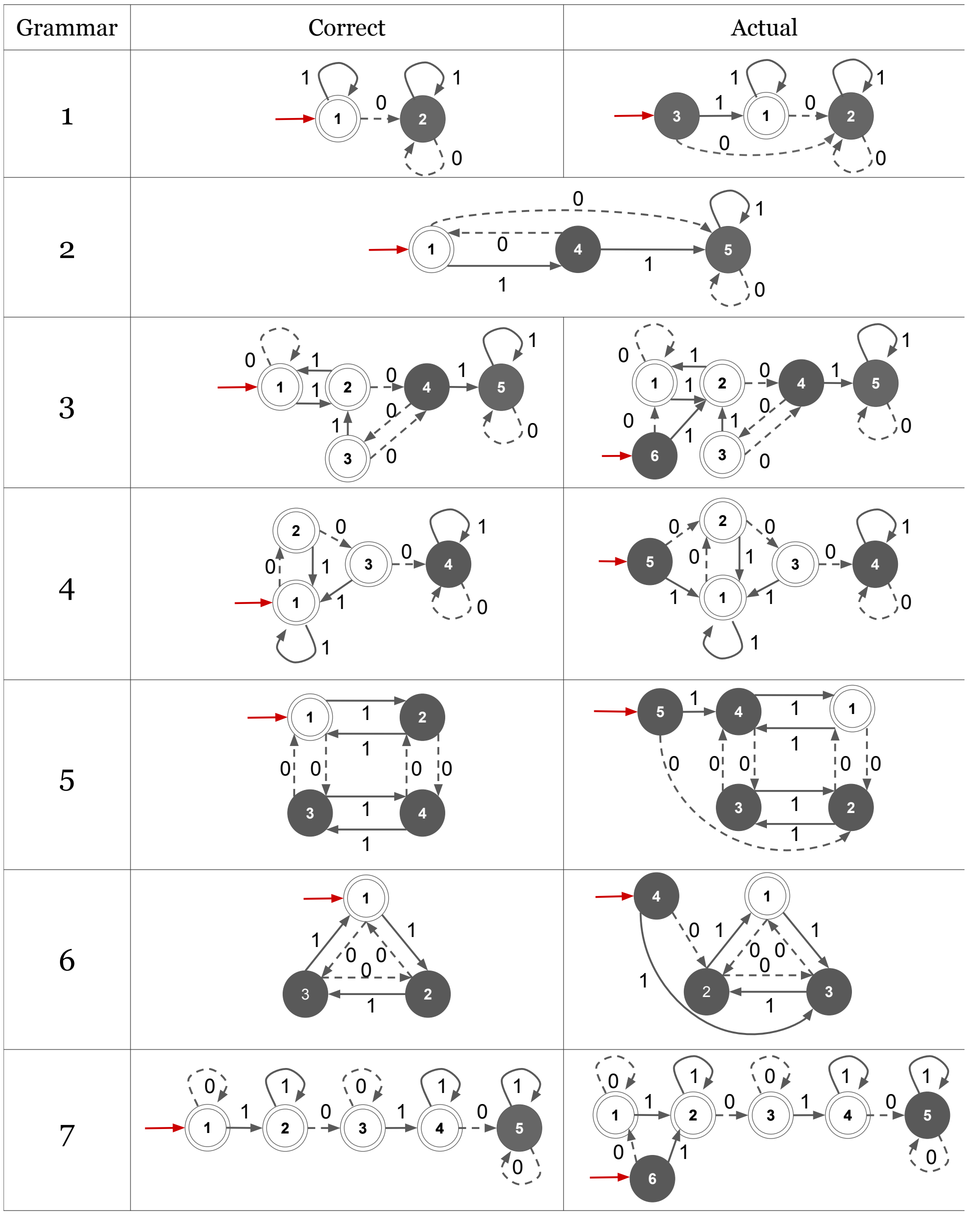
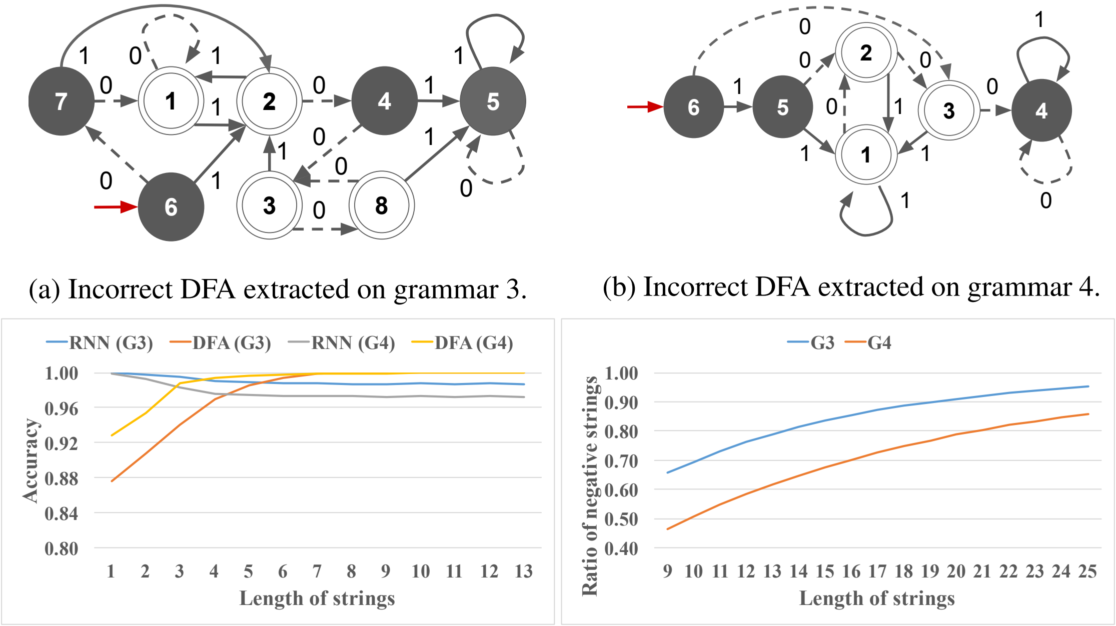


Figure 5: Visualization of ground truth DFA and extracted DFA for all grammars. Dotted lines indicate input symbol "0" and solid lines indicate input symbol "1".

periments suggest one solution of extracting a DFA from a trained RNN model, given that DFA extraction is relatively stable and a DFA can maintain its accuracy of recognizing long strings. In reality, however, it is impractical to assume that the ground truth DFA can be obtained to evaluate the extracted ones, which are possible to be incorrect. In following experiments, we empirically compare some RNNs and their "incorrectly"



(a) Incorrect DFA extracted on grammar 3.

(b) Incorrect DFA extracted on grammar 4.

(c) Comparing RNNs and DFA on testing sets comprised of longer strings.

(d) The ratio of negative strings in the complete set of strings of certain length.

Figure 6: Evaluation of small-capacity RNNs and their associated incorrect DFA for Tomita grammars 3 and 4.

extracted DFA. Here we demonstrate the results on grammar 3 and 4 due to space constraint. These grammars are selected as we observed in the experiments that the RNNs trained on these grammars are more sensitive to model capacity.

We first construct two RNNs with 9 hidden neurons and have them trained to reach 100% accuracy on data set D_{test}. Their associated incorrect DFA extracted, as shown in Figure 6a and 6b, achieve 93% and 98% accuracy on D_{test} respectively. We next evaluate these RNNs and their incorrect DFA using multiple testing sets with the number of samples fixed at 100,000 and string length varying from 20 to 200. The sampling of positive and negative strings is similar to what was described in Section.4.1.

RNN test-set performance is shown in Figure. 6c. We observe that on these test sets composed of longer strings, RNNs make more classification errors. This may due to the fact that as the string length increases, the ratio of negative strings to positive ones also increases (shown in Figure.6d). This would mean that an RNN processes more negative strings, which can be interpreted as "noisy" samples, and as a result, would generate more false positive errors. On the other hand, for the DFA associated with these RNNs, fewer and fewer mistakes are made as the number of negative strings increases. This might be the result of the fact that these incorrect DFA generate their own regular language $L_{3'}$ and $L_{4'}$ respectively, which are quite similar to the target languages L_3 and L_4. As a result, many of the negative strings rejected by the extracted DFA are also rejected by the correct DFA. As more and more negative strings are sampled, this overlapping behavior gradually dominates the testing sets. These results demonstrate that, in certain cases, it is possible to extract a DFA which does not fully represent the target RNN and yet still outperforms the RNN when processing longer sequences. Given this result, one possible path to improving an RNN's ability to handle longer sequences might lie in the exploitation of this useful DFA behavior.

5 Conclusion

We conducted a careful experimental study of the extraction of deterministic finite automata from second-order recurrent neural networks. We identified the factors that influence the reliability of the extraction process and were able to show that, despite these factors, the automata can still be stably extracted even when the neural model is trained only using short sequences. Our experiments also show that while model capacity does indeed strongly damage the neural network's ability to handle longer sequences, this hardly affects the extraction process. Furthermore, the automata extracted from low-capacity second order RNNs, in some cases, actually outperform the RNN trained model when processing sequences longer than what were seen during training. Our findings imply that one potential pathway to improving an RNN's ability to learn longer-term dependencies might be through the exploitation of the DFA's natural ability to handle infinitely long sequences and that it would be interesting to exploit transfer learning in this area. Future work will focus on comparing extracted DFAs and source RNN models on more complex/or real-world datasets consisting of long sequences such as currency exchange rates Giles et al. (2001) and others.

6 Acknowledgements

We gratefully acknowledge partial support from the College of Information Sciences and Technology.

References

Borges, R. V., d'Avila Garcez, A. S., & Lamb, L. C. (2011). Learning and representing temporal knowledge in recurrent networks. *IEEE Trans. Neural Networks*, *22*(12), 2409–2421.

Carroll, J., & Long, D. (1989). *Theory of finite automata with an introduction to formal languages*.

Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural computation*, *8*(6), 1135–1178.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, *2*(3), 113–124.

Dhingra, B., Yang, Z., Cohen, W. W., & Salakhutdinov, R. (2017). Linguistic knowledge as memory for recurrent neural networks. *arXiv preprint arXiv:1703.02620*.

Du, J., Ma, S., Wu, Y.-C., Kar, S., & Moura, J. M. (2016). Convergence analysis of distributed inference with vector-valued gaussian belief propagation. *arXiv preprint arXiv:1611.02010*.

Du, J., Zhang, S., Wu, G., Moura, J. M., & Kar, S. (2017). Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370*.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1996). Representation of finite state automata in recurrent radial basis function networks. *Machine Learning*, *23*(1), 5–32.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., & Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *Neural networks, 1991., ijcnn-91-seattle international joint conference on* (Vol. 2, pp. 273–281).

Giles, C. L., Lawrence, S., & Tsoi, A. C. (2001). Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, *44*(1-2), 161–183.

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

Giles, C. L., Sun, G.-Z., Chen, H.-H., Lee, Y.-C., & Chen, D. (1990). Higher order recurrent networks and grammatical inference. In *Advances in neural information processing systems* (pp. 380–387).

Gori, M., Maggini, M., Martinelli, E., & Soda, G. (1998). Inductive inference from noisy examples using the hybrid finite state filter. *IEEE Transactions on Neural Networks*, *9*(3), 571–575.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2013). *Introduction to automata theory, languages, and computation*.

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

Kolen, J. F. (1994). Fool's gold: Extracting finite state machines from recurrent network dynamics. In *Advances in neural information processing systems* (pp. 501–508).

Li, K., & Príncipe, J. C. (2016). The kernel adaptive autoregressive-moving-average algorithm. *IEEE Trans. Neural Netw. Learning Syst.*, *27*(2), 334–346.

Lin, T., Horne, B. G., Tiño, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Trans. Neural Networks*, *7*(6), 1329–1338.

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc.

Murdoch, W. J., & Szlam, A. (2017). Automatic rule extraction from long short term memory networks. *arXiv preprint arXiv:1702.02540*.

Omlin, C. W., & Giles, C. L. (1996a). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, *43*(6), 937–972.

Omlin, C. W., & Giles, C. L. (1996b). Constructing deterministic finite-state automata in recurrent neural networks. *Neural Computation*, *8*(4), 675–696.

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

Omlin, C. W., & Giles, C. L. (2000). Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. *Knowledge based neurocomputing*, 63–115.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine learning*, *7*(2), 227–252.

Sanfeliu, A., & Alquezar, R. (1994). Active grammatical inference: a new learning methodology. In *in shape, structure and pattern recogniton, d. dori and a. bruckstein (eds.), world scientific pub.*

Schellhammer, I., Diederich, J., Towsey, M., & Brugman, C. (1998). Knowledge extraction and recurrent neural networks: An analysis of an elman network trained on a natural language learning task. In *Proceedings of the joint conferences on new methods in language processing and computational natural language learning* (pp. 73–78).

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems* (pp. 2440–2448).

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

Tiňo, P., & Šajda, J. (1995). Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, *7*(4), 822–844.

Tomita, M. (1982a). Dynamic construction of finite automata from example using hill-climbing. *Proceedings of the Fourth Annual Cognitive Science Conference*, 105–108.

Tomita, M. (1982b). Dynamic construction of finite-state automata from examples using hill-climbing. In (pp. 105–108).

Wang, Q., Zhang, K., II, A. G. O., Xing, X., Liu, X., & Giles, C. L. (2018). A comparison of rule extraction for different recurrent neural network models and grammatical complexity. *arXiv preprint arXiv:1801.05420*.

Watrous, R. L., & Kuhn, G. M. (1992). Induction of finite-state automata using second-order recurrent networks. In *Advances in neural information processing systems* (pp. 309–317).

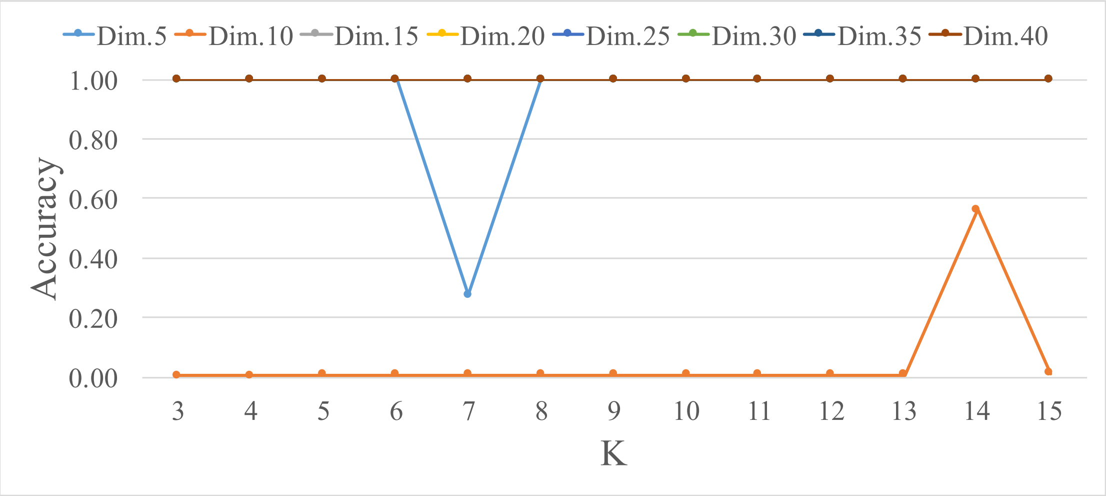
Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

Zeng, Z., Goodman, R. M., & Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, *5*(6), 976–990.

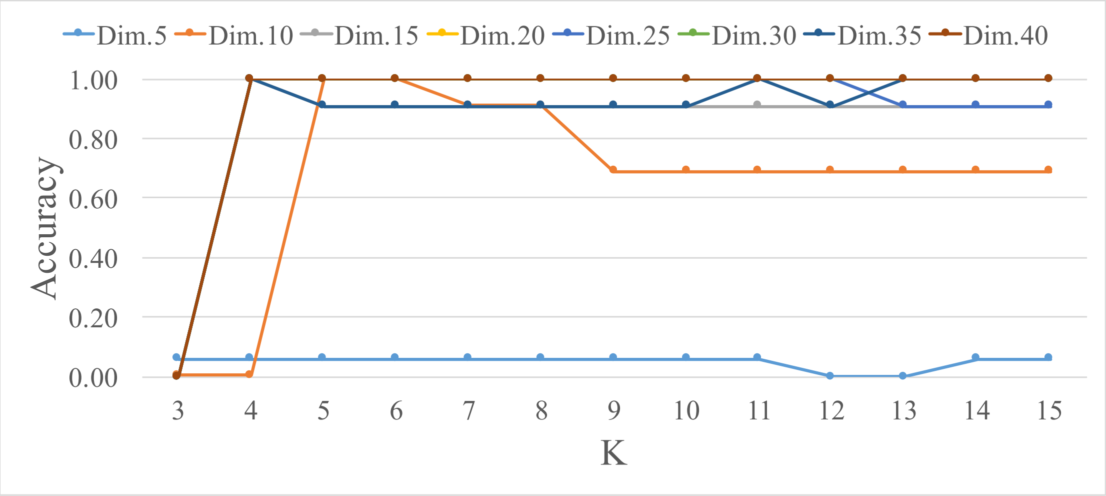
Experimental Results for All Tomita Grammars

Table 3: Influence of training time on DFA extraction and RNN performance for all Tomita grammars.

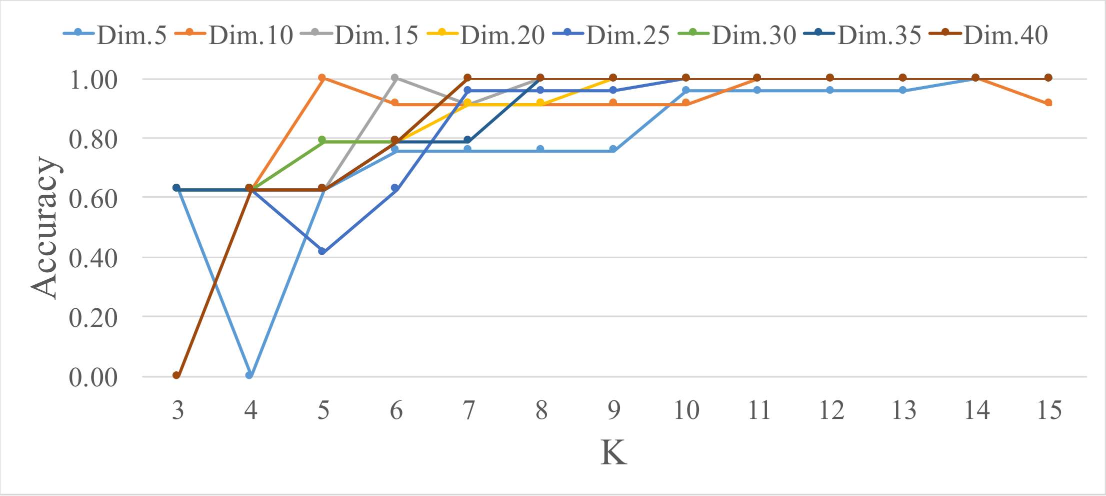
Grammar	Classification errors reached under different training epochs.							
G1	Epoch	30	60	90	120	150	180	210
	RNN(Train)	0.0	0.0	0.98	1.0	1.0	1.0	1.0
	RNN(Test)	0.0	0.0	4.2e-3	4.3e-3	4.3e-3	4.4e-3	4.4e-3
	DFA(Test)	4.0e-3	4.0e-3	1.0	1.0	1.0	1.0	1.0
G2	Epoch	100	200	300	400	500	600	700
	RNN(Train)	0.0	0.54	0.91	0.85	0.85	0.92	0.97
	RNN(Test)	0.0	0.0	2.0e-2	2.0e-2	2.0e-2	2.0e-2	2.0e-2
	DFA	2.0e-3	2.0e-3	1.0	1.0	1.0	1.0	1.0
G6	Epoch	90	120	150	180	210	240	270
	RNN(Train)	0.0	0.0	0.0	1.2e-2	1.0	1.0	1.0
	RNN(Test)	0.0	0.0	0.0	4.7e-3	1.0	1.0	1.0
	DFA	0.5	0.5	0.5	0.5	1.0	1.0	1.0
G7	Epoch	20	40	60	80	100	120	140
	RNN(Train)	0.92	0.99	0.99	1.0	1.0	1.0	1.0
	RNN(Test)	0.64	0.94	1.0	1.0	1.0	1.0	1.0
	DFA	2.0e-3	2.0e-3	1.0	1.0	1.0	1.0	1.0



(a) Influence of model capacity on DFA extraction for grammar 1.

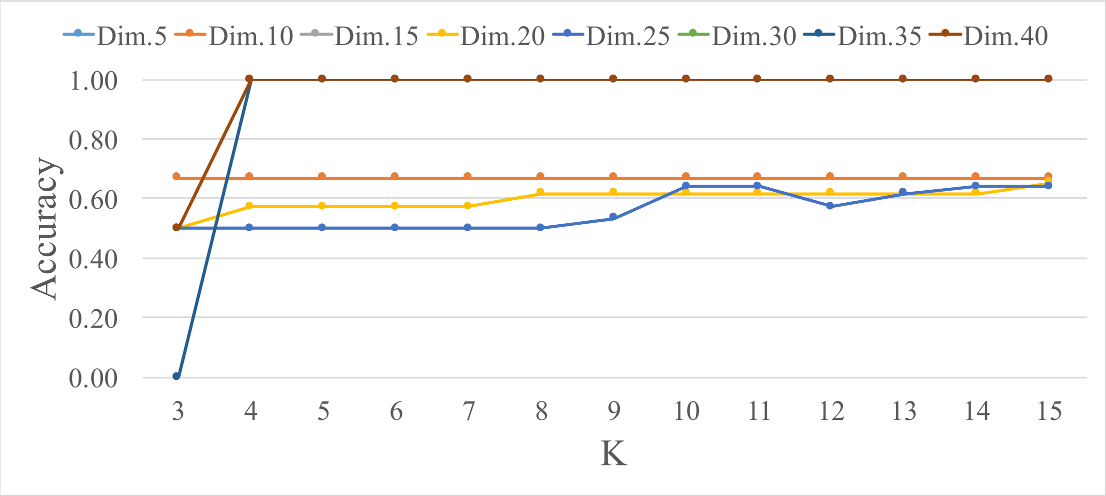


(b) Influence of model capacity on DFA extraction for grammar 2.

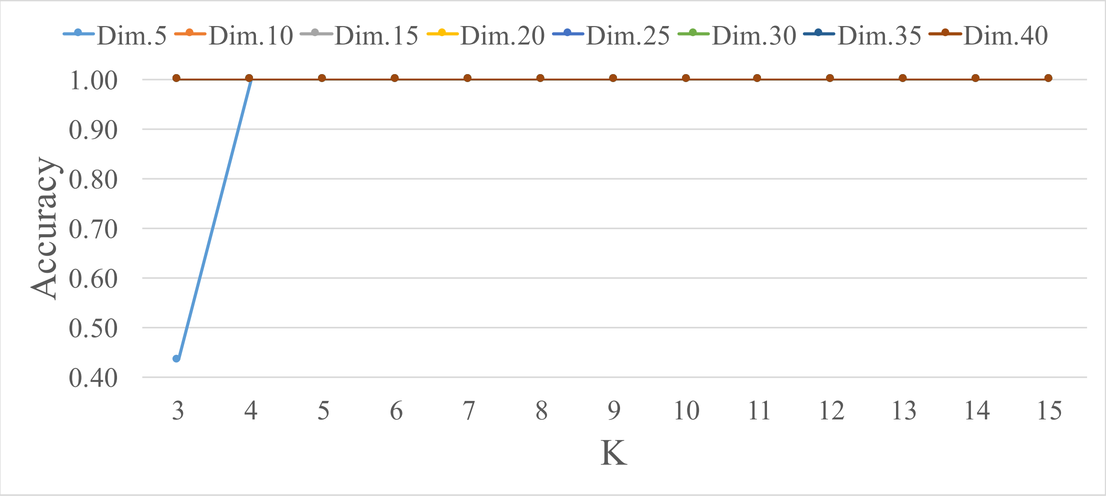


(c) Influence of model capacity on DFA extraction for grammar 4.

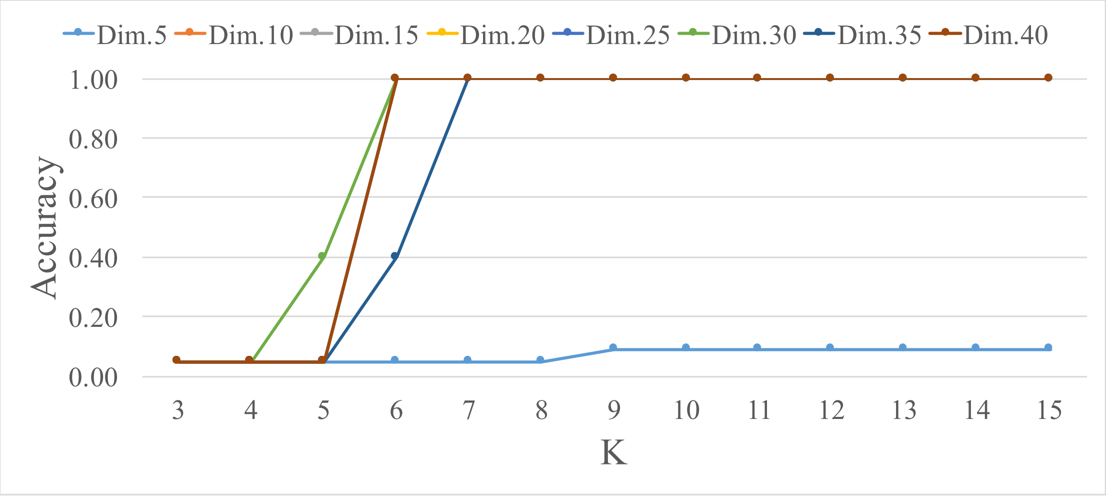
Figure 7: Influence of model capacity on DFA extraction for Tomita grammars.



(d) Influence of model capacity on DFA extraction for grammar 5.



(e) Influence of model capacity on DFA extraction for grammar 6.



(f) Influence of model capacity on DFA extraction for grammar 7.

Figure 7: Influence of model capacity on DFA extraction for Tomita grammars.