Weighted Automata Extraction from
Recurrent Neural Networks via Regression on State Spaces

Takamasa Okudono, Masaki Waga, Taro Sekiyama, Ichiro Hasuo
National Institute of Informatics & The Graduate University for Advanced Studies
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430
{tokudono, mwaga, sekiyama, hasuo}@nii.ac.jp

Abstract
We present a method to extract a weighted finite automaton (WFA) from a recurrent neural network (RNN). Our method is based on the WFA learning algorithm by Balle and Mohri, which is in turn an extension of Angluin’s classic L* algorithm. Our technical novelty is in the use of regression methods for the so-called equivalence queries, thus exploiting the internal state space of an RNN to prioritize counterexample candidates. This way we achieve a quantitative/weighted extension of the recent work by Weiss, Goldberg and Yahav that extracts DFAs. We experimentally evaluate the accuracy, expressivity and efficiency of the extracted WFAs.

1 Introduction

Background Deep neural networks (DNNs) have been successfully applied to domains such as text, speech, and image processing. Recurrent neural networks (RNNs) (Chung et al. 2014; Hochreiter and Schmidhuber 1997) is a class of DNNs equipped with the capability of processing sequential data of variable length. The great success of RNNs has been seen in, e.g., machine translation (Sutskever, Vinyals, and Le 2014), speech recognition (Zweig et al. 2017), and anomaly detection (Lv et al. 2018; Xiao et al. 2019).

While it has been experimentally shown that RNNs are a powerful tool to process, predict, and model sequential data, there are known drawbacks in RNNs such as interpretability and costly inference. A research line that attacks this challenge is automata extraction (Omlin and Giles 1996; Weiss, Goldberg, and Yahav 2018a). Focusing on RNNs’ use as acceptors (i.e., receiving an input sequence and producing a single Boolean output), these works extract a finite-state automaton from an RNN as a succinct and interpretable surrogate. Automata extraction exposes internal transition between the states of an RNN in the form of an automaton, which is then amenable to algorithmic analyses such as reachability and model checking (Baier and Katoen 2008). Automata extraction can also be seen as model compression: finite-state automata are usually more compact, and cheaper to run, than neural networks.

Extracting WFAs from RNNs Most of the existing automata extraction techniques target Boolean-output RNNs, which however excludes many applications. In sentiment analysis, it is desired to know the quantitative strength of sentiment, besides its (Boolean) existence (Chaudhuri 2019). RNNs with real values as their output are also useful in classification tasks. For example, predicting class probabilities is a key in some approaches to semi-supervised learning (Yarowsky 1995) and ensemble (Bramer 2013).

This motivates extraction of quantitative finite-state machines as abstraction of RNNs. We find the formalism of weighted finite automata (WFAs) suited for this purpose. A WFA is a finite-state machine—much like a deterministic finite automaton (DFA)—but its transitions as well as acceptance values are real numbers (instead of Booleans).

Contribution: Regression-Based WFA Extraction from RNNs Our main contribution is a procedure that takes a (real-output) RNN R, and returns a WFA A_R that abstracts R. The procedure is based on the WFA learning algorithm in (Balle and Mohri 2015), that is in turn based on the famous L* algorithm for learning DFAs (Angluin 1987). These algorithms learn automata by a series of so-called membership queries and equivalence queries. In our procedure, a membership query is implemented by an inference of the given RNN R. We iterate membership queries and use their results to construct a WFA A and to grow it.

The role of equivalence queries is to say when to stop this iteration: it asks if A and R are “equivalent,” that is, if the WFA A obtained so far is comprehensive enough to cover all the possible behaviors of R. This is not possible in general—RNNs are more expressive than WFAs—therefore we inevitably resort to an approximate method. Our technical novelty lies in the method for answering equivalence queries; notably it uses a regression method—e.g., the Gaussian process regression (GPR) and the kernel ridge regression (KRR)—for abstraction of the state space of R.

We conducted experiments to evaluate the effectiveness of our approach. In particular, we are concerned with the following questions: 1) how similar the behavior of the extracted WFA A_R and that of the original RNN R are; 2) how applicable our method is to an RNN that is a more expressive model than WFAs; and 3) how efficient the in-
ference of the WFA $A_R$ is, when compared with the inference of $R$. The experiments we designed for the questions 1) and 3) are with RNNs trained using randomly generated WFs. The results show, on the question 1), that the WFs extracted by our method approximate the original RNNs accurately. This is especially so when compared with a baseline algorithm (a straightforward adaptation of Balle and Mohri (2015)). On the question 3), the inference of the extracted WFs are about 1300 times faster than that of the original RNNs. On the question 2), we devised an RNN that models a weighted variant of a (non-regular) language of balanced parentheses. Although this weighted language is beyond WFs’ expressivity, we found that our method extracts WFs that successfully approximate the RNN up-to a certain depth bound.

The paper is organized as follows. Angluin’s $L^*$ algorithm and its weighted adaptation are recalled in §2. Our WFA extraction procedure is described in §3 focusing on our novelty, namely the regression-based procedure for answering equivalence queries. Comparison with the DFA extraction by Weiss, Goldberg, and Yahav (2018a) is given there, too. In §4 we discuss our experiment results.

**Potential Applications** A major potential application of WFA extraction is to analyze an RNN $R$ and its balanced parentheses. Although this weighted language is beyond WFs’ expressivity, we found that our method extracts WFs whose example is an RNN.

Their method does not use equivalence queries; in contrast, we exploit the internal state space of an RNN to approximately answer equivalence queries.

Deep Stellar (Du et al. 2019) extracts Markov chains from RNNs, and uses them for coverage-based testing and adversarial sample detection. Their extraction, differently from our $L^*$-like method, uses profiling from the training data and discrete abstraction of the state space.

Wang and Niepert (2019) propose a new RNN architecture that makes it easier to extract a DFA from a trained model. To apply their method, one has to modify the structure of an RNN before training, while our method does not need any special structure to RNNs and can be applied to already trained RNNs.

Schwartz, Thomson, and Smith (2018) introduce a neural network architecture that can represent (restricted forms of) CNNs and RNNs. WFs could also be expressed by their architecture, but extraction of automata is out of their interest.

A major approach to optimizing neural networks is by compression: model pruning (Han et al. 2015), quantization (Gupta et al. 2015), and distillation (Bucila, Caruana, and Niculescu-Mizil 2006). Combination and comparison with these techniques is interesting future work.

## 2 Preliminaries

We fix a finite alphabet $\Sigma$. The set of (finite-length) words over $\Sigma$ is $\Sigma^*$. The empty word (of length 0) is denoted by $\varepsilon$. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. We recall basic notions on WFs. See (Droste, Kucih, and Vogler 2009) for details.

**Definition 1 (WFA).** A weighted finite automaton (WFA) over $\Sigma$ is a quadruple $A = (Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma \in \Sigma})$. Here $Q_A$ is a finite set of states; $\alpha_A, \beta_A$ are row vectors of size $|Q_A|$ (called the initial and final vectors); and $A_\sigma$ is a transition matrix of $\sigma$, given for each $\sigma \in \Sigma$. For each $\sigma \in \Sigma$, $A_\sigma$ is a matrix of size $|Q_A| \times |Q_A|$.

**Definition 2 (configuration of a WFA).** Let $A$ be the WFA in Def. 1. A configuration of $A$ is a row vector $x \in \mathbb{R}^{Q_A}$. For a word $w = \sigma_1 \sigma_2 \ldots \sigma_n \in \Sigma^*$ (where $\sigma_i \in \Sigma$), the configuration of $A$ at $w$ is defined by $\delta_A(w) = \alpha_A \cdot (\prod_{i=1}^{n} A_{\sigma_i})$.

Obviously $\delta_A(w) \in \mathbb{R}^{Q_A}$ is a row vector of size $|Q_A|$; it records the weight at each state $q \in Q_A$ after reading $w$.

**Definition 3 (weight $f_A(w)$ of a word in a WFA).** Let a WFA $A$ and a word $w = \sigma_1 \ldots \sigma_n$ be as in Def. 2. The weight of $w$ in $A$ is given by $f_A(w) = \alpha_A \cdot (\prod_{i=1}^{n} A_{\sigma_i}) \cdot \beta_A$, multiplying the final vector to the configuration at $w$.

**Example 4.** Let $\Sigma = \{a, b\}$, $Q_A = \{q_1, q_2, q_3\}$, $\alpha_A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & 0 \\ 0 & 4 & 0 \end{pmatrix}$, and $\beta_A = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 3 & 0 \\ -2 & 4 & 0 \end{pmatrix}$. For the WFA $A = (Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma \in \Sigma})$ over $\Sigma$, and $w = ba$, the config-
The configuration $\delta_R(w)$ of the RNN $R$ at a word $w$ is defined by $\delta_R(w) = g_R^w(\alpha_R, w)$. The output $f_R(w) \in \mathbb{R}$, of $R$ for the input $w$, is defined by $f_R(w) = \beta_R(\delta_R(w))$.

### 2.1 Angluin’s L* Algorithm

Angluin’s L*-algorithm learns a given DFA $B$ by a series of membership and equivalence queries. We sketch the algorithm; see (Angluin 1987) for details. Its outline is in Fig. 2.

A membership query is a black-box observation of the DFA $B$: it feeds $B$ with a word $w \in \Sigma^*$; and obtains $f_B(w) \in \{tt, ff\}$, i.e., whether $w$ is accepted by $B$.

The core of the algorithm is to construct the observation table $T$; see Fig. 3a. The table has words as the row and column labels; its entries are either tt or ff. The row labels are called access words; the column labels are test words. We let $A, T$ stand for the sets of access and test words. The entry of $T$ at row $u \in A$ and column $v \in T$ is given by $f_B(u, v)$—a value that we can obtain from a suitable membership query.

Therefore we extend a table $T$ by a series of membership queries. We do so until $T$ becomes closed; this is the top loop in Fig. 2. A table $T$ is closed if, for any access word $u \in A$ and $\sigma \in \Sigma$, there is an access word $u' \in A$ such that $f_B(u, \sigma v) = f_B(u', v)$ for each test word $v \in T$. (1)

The closedness condition essentially says that the role of the extended word $u \sigma v$ is already covered by some existing word $u' \in A$. The notion of “role” here, formalized in (1), is a restriction of the well-known Myhill–Nerode relation, from all words $w \in \Sigma^*$ to $v \in T$.

A closed table $T$ induces a DFA $A_T$ (Fig. 2), much like in the Myhill–Nerode theorem. We note that the resulting DFA $A_T$ is necessarily minimized. The DFA $A_T$ undergoes an equivalence query that asks if $A_T \cong B$; an equivalence query is answered with a counterexample—i.e., $w \in \Sigma^*$ such that $f_{A_T}(w) \neq f_B(w)$—if $A_T \not\cong B$.

---

**Figure 1:** An illustration of WFA $A$ in Example 4. In a state label “q/m/n”, $q$ is a state name and $m$ and $n$ are the initial and final values at $q$, respectively. In the label “$\sigma,p$” of the transition from $q_i$ to $q_j$, $p$ is $A_\sigma[i,j]$, where $\sigma \in \Sigma$ and $A_\sigma[i,j]$ is the entry of $A_\sigma$ at row $i$ and column $j$.

**Figure 2:** An outline of Angluin’s L* algorithm. The target DFA is $B$; a table $T$ gets gradually extended, yielding a DFA $A_T$ when it is closed. See also Fig. 3a.

| $A \setminus T$ | $\varepsilon$ | 0 | 1 |
|-----------------|------------|---|---|
| $\varepsilon$   | tt         | tt| tt|
| 0               | ff         | ff| ff|
| 011             | tt         | ff| ff|
| 001             | ff         | ff| ff|

(a) A table $T$ for DFA learning (b) A table $T$ for WFA learning

**Figure 3:** Observation tables for L*-style algorithms

The configuration $\delta_R(w)$ of the RNN $R$ at a word $w$ is defined by $\delta_R(w) = g_R^w(\alpha_R, w)$. The output $f_R(w) \in \mathbb{R}$, of $R$ for the input $w$, is defined by $f_R(w) = \beta_R(\delta_R(w))$.
The L* algorithm is a deterministic learning algorithm (at least in its original form), unlike many recent learning algorithms that are statistical. The greatest challenge in practical use of the L* algorithm is to answer equivalence queries. When B is a finite automaton that generates a regular language, there is a complete algorithm for deciding the language equivalence $A_T \cong B$. However, if we use a more expressive model in place of a DFA B, checking $A_T \cong B$ becomes a nontrivial task.

### 2.2 L* Algorithm for WFA Learning

The classic L* algorithm for learning DFAs has seen a weighted extension (Balle and Mohri 2015): it learns a WFA $B$, again via a series of membership and equivalence queries. The overall structure of the WFA learning algorithm stays the same as in Fig. 2; here we highlight major differences.

Firstly, the entries of an observation table $T$ are now real numbers, reflecting the fact that the value $f_B(uv)$ for a WFA $B$ is in $\mathbb{R}$ instead of in $\{\text{tt}, \text{ff}\}$ (see Def. 3). An example of an observation table is given in Fig. 3b.

Secondly, the notion of closedness is adapted to the weighted (i.e., linear-algebraic) setting, as follows. A table $T$ is closed if, for any access word $u \in A$ and $\sigma \in \Sigma$, the vector $(f_B(u\sigma v))_{v \in T} \in \mathbb{R}^{|T|}$ can be expressed as a linear combination of the vectors in $\{(f_B(u'v'))_{v' \in T} | u' \in A\}$. Note that the vector $(f_B(u'v'))_{v' \in T}^T$ in the latter set is precisely the row vector in $T$ at row $u'$. For example, the table $T$ in Fig. 3b is obviously closed, since the three row vectors are linearly independent and thus span the whole $\mathbb{R}^3$. The above definition of closedness comes natural in view of Def. 2. For a WFA, a configuration (during its execution) is not a single state, but a weighted superposition $x \in \mathbb{R}^{Q_A}$ of states. The closedness condition asserts that the role of $u\sigma$ is covered by a suitable superposition of words $u' \in A$. The construction of the WFA $A_T$ from a closed table $T$ (see Fig. 2) reflects this intuition. See (Balle and Mohri 2015). We note that the resulting $A_T$ is minimal, much like in §2.1.

In the literature (Balle and Mohri 2015), an observation table $T$ is presented as a so-called Hankel matrix. This opens the way to further extensions of the method, such as an approximate learning algorithm via the singular-value decomposition (SVD).

### 3 WFA Extraction from an RNN

We present our main contribution, namely a procedure that extracts a WFA from a given RNN. After briefly describing its outline (that is the same as Fig. 2), we focus on the greatest challenge of answering equivalence queries.

#### Algorithm 1 Answering equivalence queries

1. **procedure** `ANS-EQQ`  
   **Input**: RNN $R = (\alpha_R, \beta_R, g_R)$, WFA $A = (Q_A, \sigma_A, \beta_A, (A_\sigma)_{\sigma \in \Sigma})$, error tolerance $\epsilon > 0$ and concentration threshold $M \in \mathbb{N}$  
   **Output**: a counterexample, or Equivalent  
2. Initialize $p: \mathbb{R}^d \to \mathbb{R}^{Q_A}$ so that $p(\delta_B(\epsilon)) = \delta_A(\epsilon)$  
3. queue $\leftarrow (\epsilon); \text{visited} \leftarrow \emptyset$  
4. while queue is non-empty do  
5. $h \leftarrow \text{pop(queue)}$  
6. if $|f_B(h) - f_A(h)| \geq \epsilon$ then  
   return $h$  
7. result $\leftarrow \text{CONSISTENT?}(h, \text{visited}, p)$  
8. if result $= \text{NG}$ then  
   learn $p$ by regression, so that $p(\delta_B(h')) = \delta_A(h')$ holds for all $h' \in \text{visited} \cup \{h\}$  
9. visited $\leftarrow \text{visited} \cup \{h\}$  
10. visited $\leftarrow p(\delta_B(\text{visited}))$  
11. $\#m$ $\leftarrow \{|x \in \text{visited} | x \simeq_A p(\delta_B(h))\}$  
12. if $\#m \leq M$ then  
   $pr$ $\leftarrow \min_{h' \in \text{visited} \cap \{h\}} d(p(\delta_B(h)), (p(\delta_B(h'))) \geq d$ is the Euclidean distance  
13. $\text{push} h\sigma$ to queue with priority $pr$ for $\sigma \in \Sigma$  
14. return Equivalent
in Fig. 2). It calls the auxiliary procedure Consistent?, which decides if we refine the current configuration abstraction function \(p: \mathbb{R}^d \rightarrow \mathbb{R}^{Q_A}\) in Fig. 4 (Line 10).

**Best-First Search for a Counterexample** The procedure ANS-EQQ is essentially a best-first search for a counterexample, that is, a word \(h \in \Sigma^*\) such that the difference of the output values \(f_R(h)\) from the RNN and \(f_A(h)\) from the WFA is larger than error tolerance \(e(>0)\). We first outline ANS-EQQ and then go into the technical detail.

We manage counterexample candidates by the priority queue. If all the candidates are not a counterexample, ANS-EQQ returns them. Note that we add the output values returned by ANS-EQQ.

We search for a counterexample in the main loop starting from Line 4. Let \(h\) be a word popped from queue, that is, the candidate most likely to be a proper counterexample among the words in the queue. If \(h\) is a counterexample, ANS-EQQ returns it (Lines 6–7). Otherwise, after refining the configuration abstraction function \(p\) (Lines 8–10), new candidates \(h\sigma\), the extension of \(h\) with character \(\sigma\), are pushed to queue more likely to be a counterexample. The already investigated words are in the set visited. The queue initially contains only the empty word \(\varepsilon\) (Line 3).

We use \(\delta\) for configuration abstraction function \(p\) to define the relation \(\simeq_A\) in the following sense. The existence of \(h'\) satisfying the condition (3) approximates the violation of the property (2) in the following sense. If there is a word \(h'\) satisfying the first part of the condition (3), \(p\) to be refined because we find the property (2) violated with \(h'\). The second part of the condition (3) means that \(h'\) seems to behave similarly to \(h\) according to the configuration abstraction function \(p\). We expect the second part to prevent \(p\) from being refined with outliers because the neighborhoods of the words used for refining \(p\) have been investigated twice or more.

**Equivalence Relation \(\simeq_A\)** For a given WFA \(A\), we define the relation \(\simeq_A\) in the configuration space \(\mathbb{R}^{Q_A}\) by

\[
\bar{x} \simeq_A \bar{y} \iff \sum_{i=1}^{|Q_A|} \beta_i^2 (x_i - y_i)^2 < \frac{\varepsilon^2}{|Q_A|},
\]

where \(\beta\) is the final vector of the WFA \(A\). It satisfies the following.

1. If \(x \simeq_A y\) holds, the difference of the output values for the configurations \(x\) and \(y\) of the WFA \(A\) is smaller than the error tolerance \(e\), that is, \(|(x - y) \cdot \beta| < \varepsilon\).
2. If the \(i\)-th element of the final vector \(\beta\) becomes large, the neighborhood \(\{ y \in \mathbb{R}^{Q_A} \mid x \simeq_A y \}\) of \(x\) shrinks in the direction of the \(i\)-th axis.
3. The neighborhood defined by \(\simeq_A\) is an ellipsoid—a reasonable variant of an open ball as a neighborhood.

**A Heuristic for Equivalence Checking of a WFA and an RNN** Although the best-first search above works well, we introduce an additional heuristic to improve the run-time performance of our algorithm. The heuristic deems \(R\) and \(A\) to be equivalent if word \(h\) previously popped from queue is so long that it is impossible to occur in the training of the RNN. This heuristic is based on the expectation that, when \(e\) is unlikely to be a counterexample, all possible words included are unlikely to be a counterexample, and so \(R\) and \(A\) are considered equivalent. This heuristic is adopted immediately after popping \(h\) (Line 5), as follows: we suppose that the maximum length \(L\) of possible words is given; and, if the length of \(h\) is larger than \(L\), ANS-EQQ returns Equivalent. We confirm the usefulness of this heuristic in §4 empirically.

**Termination of the Procedure** Algorithm 1 does not always terminate in a finite amount of time. If the procedure does not find any counterexample at Line 7 and the points \(p(\delta_R(\text{visited}))\) are so scattered in the configuration space that the value \(#\text{un}\) at Line 13 is always small, words are always pushed to queue at Line 16. In that case, the condition to exit the main loop at Line 4 is never satisfied.

### 3.3 Comparison with Weiss et al., 2018

Our WFA extraction method can be seen as a weighted extension of the \(L^*\)-based procedure (Weiss, Goldberg, and...
Parameters are fixed as follows: error tolerance $\epsilon = 0.05$. We implemented our method in BHFS, the baseline algorithm.

### 4.1 RQ1: Extraction from RNNs Modeling WFAs

We report the results after presenting the details of the baseline, how to train RNNs, and how to evaluate the two algorithms.

#### 4.1.1 Experimental Setting

To do so, we first train RNNs using randomly generated WFAs; we call those WFAs the *origins*. Then, we evaluate our algorithm compared with a baseline "Total" row). More generally, we can find the trend that, as $n$ is larger, more counterexample candidates are investigated.

#### 4.1.2 Evaluation

A simple way to sample words $w$ of $T$ is by the uniform distribution. With $T$ covering the input space of the WFA $A^*$ uniformly, we expect the resulting RNN $R$ to inherit properties of $A^*$ well. The top table in Table 1 reports the results in this "more WFA-like" setting.

#### 4.1.3 Results and Discussions

In the experiments in Table 1, we considered 8 configurations for generating the original WFA $A^*$ (the leftmost column). The unit of MSEs are --4—given also that the outputs of the original WFAs $A^*$ are normalized to $[0, 1]$, we can say that the MSEs are small enough.

In the top table in Table 1 (the "more WFA-like" setting), BFS(5000) and RGR(5) achieved the first- and second-best performance in terms of accuracy, respectively (see the "Total" row). More generally, we can find the trend that, as an extraction runs longer, it performs better. We conjecture its reason as follows. Recall that all the RNNs are trained on words sampled from the uniform distribution. This means...
Table 1: Experiment results, where we extracted a WFA $A$ from an RNN $R$ that is trained to mimic the original WFA $A^\star$. In each cell “n/m”, “n” denotes the average of MSEs between $A$ and $R$ (the unit is $10^{-4}$), taken over five random WFAs $A^\star$ of the designated alphabet size $|\Sigma|$ and the state-space size $|Q_{A^\star}|$. “m” denotes the average running time (the unit is second). The “Total” row describes the average over all the experiment settings. The highlighted cell designates the best performer in terms of errors. Timeout was set at 10,000 sec. RGR(2–5) are our regression-based methods; BFS(500–5000) are the baseline. BFS(5000) is added to compare the accuracy when the running time is much longer.

| $(|\Sigma|, |Q_{A^\star}|)$ | RGR(2) | RGR(5) | BFS(500) | BFS(1000) | BFS(2000) | BFS(5000) |
|------------------------|--------|--------|----------|-----------|-----------|-----------|
| (4, 10)                | 2.17 / 286 | 2.39 / 338 | 26.87 / 165 | 9.77 / 279 | 4.36 / 345 | 4.07 / 716 |
| (6, 10)                | 2.45 / 1787 | 2.54 / 1302 | 6.99 / 386 | 4.48 / 641 | 4.08 / 1218 | 3.15 / 1410 |
| (10, 10)               | 4.68 / 7462 | 4.46 / 5311 | 22.5 / 928 | 11.9 / 1562 | 5.90 / 3521 | 4.55 / 3638 |
| (15, 10)               | 5.62 / 8941 | 5.78 / 8564 | 21.2 / 2155 | 10.6 / 4750 | 7.87 / 5692 | 5.71 / 7344 |
| (20, 10)               | 3.70 / 7610 | 3.79 / 7799 | 6.24 / 2465 | 10.1 / 2188 | 6.13 / 3106 | 3.70 / 5729 |
| (15, 15)               | 7.34 / 9596 | 5.52 / 10000 | 13.5 / 3227 | 8.01 / 6765 | 6.07 / 7916 | 5.98 / 8911 |
| (20, 15)               | 8.44 / 18000 | 5.58 / 9981 | 16.3 / 2675 | 9.24 / 4850 | 7.28 / 5135 | 9.88 / 7204 |
| (25, 20)               | 9.16 / 7344 | 5.15 / 7857 | 13.7 / 2224 | 7.26 / 3823 | 6.60 / 5744 | 4.96 / 5674 |
| **Total**              | 5.45 / 6628 | 4.49 / 6994 | 15.97 / 1778 | 8.95 / 3107 | 6.04 / 4110 | 5.25 / 5078 |

Results for “more WFA-like” RNNs $R$, i.e., those RNNs $R$ which are trained from WFAs $A^\star$ using a uniformly sampled data set $T \subseteq \Sigma^*$ for simplicity, too; the weight threshold for showing transitions is $-2.54\times 10^{-2}$.

| $(|\Sigma|, |Q_{A^\star}|)$ | RGR(2) | RGR(5) | BFS(500) | BFS(1000) | BFS(2000) | BFS(5000) |
|------------------------|--------|--------|----------|-----------|-----------|-----------|
| (4, 10)                | 7.37 / 696 | 7.07 / 1135 | 15.07 / 199 | 7.96 / 428 | 6.62 / 630 | 6.61 / 762 |
| (6, 10)                | 4.92 / 1442 | 7.43 / 1247 | 1.46 / 552 | 6.95 / 660 | 5.90 / 1217 | 8.78 / 1557 |
| (10, 10)               | 5.02 / 5536 | 4.29 / 9591 | 1.72 / 1117 | 11.0 / 1738 | 4.77 / 2635 | 3.52 / 3926 |
| (15, 10)               | 7.15 / 697 | 4.35 / 3815 | 19.4 / 1552 | 13.8 / 3271 | 16.8 / 3209 | 8.57 / 5293 |
| (20, 10)               | 6.98 / 4697 | 8.06 / 6704 | 18.6 / 1465 | 11.8 / 2046 | 12.7 / 2851 | 9.03 / 4259 |
| (15, 15)               | 5.97 / 8747 | 6.77 / 8882 | 23.3 / 2359 | 11.2 / 4668 | 9.88 / 6186 | 6.24 / 7557 |
| (20, 15)               | 5.78 / 8325 | 8.71 / 7546 | 16.6 / 2874 | 7.31 / 4380 | 9.92 / 6015 | 9.89 / 7110 |
| (25, 20)               | 6.02 / 7550 | 6.90 / 6015 | 19.01 / 1502 | 11.7 / 2532 | 9.92 / 3344 | 8.08 / 4466 |

RGR(2) performed well for smaller alphabets ($|\Sigma| \in \{4, 6, 10\}$) but not so when $|\Sigma| = 15$. The role of the parameter $M$ in RGR($M$) (i.e., in Algorithm 1) is a threshold to control how many words configuration regions of a WFA are investigated with. Thus, we conjecture that the use of too small $M$ limits the input space to be investigated excessively, which is more critical as the input space is larger, eventually biasing the counterexamples $h$ (in Algorithm 1), though the RNNs are trained on the uniform distribution, and making refinement of WFAs less effective.

In the bottom table in Table 1 (the “realistic” setting), RGR(2) performs significantly better than the other (and the best among all the procedures) in terms of accuracy. This is the case even for a large alphabet ($|\Sigma| = 15$). This indicates that, in the cases that an RNN is trained with a nonuniform dataset, making the investigated input space larger by big $M$ could even degrade the accuracy performance. A possible reason for this degradation is as follows. Some words (such as aba) are prohibited in the sample set $T$, and the behaviors of the RNN $R$ for those prohibited words are unexpected. Therefore, those prohibited words should not be useful to refine a WFA. The use of small $M$ could prevent such meaningless (or even harmful) counterexamples $h$ from being investigated. This discussion raises another question: how can we find an optimal $M$? We leave it as future work.

Let us briefly discuss the sizes of the extracted WFAs. The general trend is that the extracted WFAs $A$ have a few times greater number of states than the original WFAs $A^\star$ used in training $R$. For example, in the setting of the top table in Table 1, for $|\Sigma| = 15$ and $|Q_{A^\star}| = 20$, the average number of the states of the extracted $A$ was 38.2.

Figure 5: The WFAs extracted by RGR(5) (left) and RGR(15) (right). In a state label “$q/m/n$”, $q$ is the state name, $m$ is the initial value and $n$ is the final value. Bigger values are underlined; other values are negligibly small. The dotted and solid edges are labelled with “(“ and “)”, respectively; the edges with labels 0, 1, . . . , 9 are omitted. The edge weights are omitted for simplicity, too; the weight threshold for showing transitions is 0.01. Appendix B.2 of (Okudono et al. 2019).
4.2 RQ2: expressivity beyond WFAs

We conducted experiments to examine how well our method works for RNNs modeling languages that cannot be expressed by any WFA. Specifically, we used an RNN that models the following function \( wparen: \Sigma^* \to \{0, 1\}: \Sigma = \{(,), 0, 1, \ldots, 9\}, \ wparen(w) = 1 - (1/2)^N \) if all the parentheses in \( w \) are balanced (here \( N \) is the depth of the deepest balanced parentheses in \( w \)); and \( wparen(w) = 0 \) otherwise. This \( wparen \) is a weighted variant of a (non-regular) language of balanced parentheses. For instance, \( \wparen(((3)(7)))^* = 0, \ wparen(((3)(7)))^* = 1 - (1/2)^2 = 3/4, \) and \( wparen((a)(b)(c))^* = 1/2. \)

We trained an RNN \( R \) as follows. We generated datasets \( T_{\text{good}} \) and \( T_{\text{bad}}, \) and trained an RNN \( R \) on the set of input-output pairs of \( w \in T_{\text{good}} \cup T_{\text{bad}} \) and \( wparen(w). \) The dataset \( T_{\text{good}} \) consists of randomly generated words where all the parentheses are balanced; \( T_{\text{bad}} \) is constructed similarly, except that we apply suitable mutation to each word, which most likely makes the parentheses unbalanced. See Appendix B.1 of (Okudono et al. 2019) for details.

Fig. 5 shows the WFAs extracted from \( R. \) Remarkable observations are as follows.

- The shapes of the WFAs—obtained by ignoring clearly negligible weights—give rise to NFAs that recognize balanced parentheses up-to a certain depth.

- As the parameter \( M \) in RGR(\( M \)) grows, the recognizable depth bound grows: depth one with RGR(5); and depth two with RGR(15).

We believe these observations demonstrate important features, as well as limitations, of our method. Overall, the extracted WFAs expose interpretable structures hidden in an RNN: the NFA structures in Fig. 5 are human-interpretable (they are easily seen to encode balancedness) and machine-processable (such as determination and minimization). It is also suggested that the parameter \( M \) gives us flexibility in the trade-off of extraction cost and accuracy. At the same time, we can only obtain a truncated and regularized version of the RNN structure—this is an inevitable limitation as long as we use the formalism of WFAs.

We also note that, in each of the two extracted WFAs, the transition matrices \( A_{\sigma} \) are similar for all \( \sigma \in \{0, 1, \ldots, 9\} \) (the entries at the same position have the same order). This is as expected, too, since the function \( wparen \) does not distinguish the characters 0, 1, \ldots, 9.

4.3 RQ3: Accelerating Inference Time

We conducted experiments about inference time, comparing the original RNNs \( R \) and the WFAs \( A \) that we extracted from \( R. \) We used the same RNNs \( R \) and WFAs \( A \) as in §4.1, where the latter are extracted using RGR(2–5) and BFS(500–5000). We note that the inference of RNNs utilizes GPUs while that of WFAs is solely done by CPUs.

We observed that the inference time of the extracted WFAs \( A \) was about 1,300 times faster than the target RNNs \( R, \) taking the average over different settings (Appendix B.3 of (Okudono et al. 2019)). This demonstrates the potential use of the extracted WFAs as a computationally cheaper surrogate for RNNs. We attribute the acceleration to the following: 1) WFAs use only linear computation while RNNs involve nonlinear ones; and 2) overall, extracted WFAs are smaller in size. Provided that the accuracy of extracted WFAs can be high (as we observed in §4.1), we believe the replacement of RNNs by WFAs is a viable option in some application scenarios.

5 Conclusions and Future Work

We proposed a method that extracts a WFA from an RNN, focusing on RNNs that take a word \( w \in \Sigma^* \) and return a real value. We used regression to investigate and abstract the internal states of RNNs. We experimentally evaluated our method, comparing its performance with a baseline whose equivalence queries are based on simple breadth-first search.

One future work is a detailed comparison with other methods for model compression. Another future work is to use machine learning methods to find a counterexample in the equivalence query, such as reinforcement learning (Mnih et al. 2015) adversarial attacks (Papernot et al. 2016), and acquisition functions of GPR. Finally, we need a means to optimize parameter \( M \) of our method for a specific problem. It may also be helpful to extend our method so that the investigated words can be restricted to a fixed language \( L \subset \Sigma^*; \) if \( L \) identifies the input space of the training dataset for RNNs, we could avoid investigating the input space on which the RNNs are not trained, and therefore we could seek only “meaningful” counterexamples even in using large \( M. \)

6 Acknowledgments

Thanks are due to Mahito Sugiyama and the anonymous reviewers of AAAI for a lot of useful comments. This work is partially supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JSPS KAKENHI Grant Numbers JP15KT0012, JP18J22498, JP19K20247, JP19K22842, and JST-Mirai Program Grant Number JPMJMI18B, Japan.

References

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

Ayache, S.; Eyraud, R.; and Goudian, N. 2018. Explaining black boxes on sequential data using weighted automata. In Unold, O.; Dyrka, W.; and Wieczorek, W., eds., Proc. ICGI 2018, volume 93 of Proceedings of Machine Learning Research, 81–103. PMLR.

Baier, C., and Katoen, J.-P. 2008. Principles of Model Checking. The MIT Press.

Balle, B., and Mohri, M. 2015. Learning weighted automata. In Maletti, A., ed., Proc. CAI 2015, volume 9270 of Lecture Notes in Computer Science, 1–21. Springer.

Balle, B.; Gourdeau, P.; and Panangaden, P. 2017. Bisimulation metrics for weighted automata. In Chatzigiannakis, I.; Indyk, P.; Kuhn, F.; and Muscholl, A., eds., Proc. ICALP 2017, volume 80 of LIPIcs, 103:1–103:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.
Learning Research works via regression on state spaces. Okudono, T.; Waga, M.; Sekiyama, T.; and Hasuo, I. 2019. Deep reinforcement learning. Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Omlin, C. W.; and Giles, C. L. 1996. Extraction of rules and Russo, A., eds., 2019. Deep learning systems. In Dumas, M.; Pfahl, D.; Apel, S.; and Hassabis, D. 2015. Human-level control through Hierarchical Deep Learning Networks. Springer Briefs in Computer Science. Springer. Chung, J.; Gülçehre, Ç.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR abs/1412.3555. Droste, M.; Kuich, W.; and Vogler, H. 2009. Handbook of Weighted Automata. Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg. Du, X.; Xie, X.; Li, Y.; Ma, L.; Liu, Y.; and Zhao, J. 2019. Deepstellar: model-based quantitative analysis of stateful deep learning systems. In Dumas, M.; Pfah, D.; Apel, S.; and Russo, A., eds., Proc. ESEC/FSE 2019, 477–487. ACM. Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep learning with limited numerical precision. In Proc. ICML 2015, 1737–1746. Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both weights and connections for efficient neural network. In Proc. NIPS 2015, 1135–1143. Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. Neural Computation 9(8):1735–1780. Lv, S.; Wang, J.; Yang, Y.; and Liu, J. 2018. Intrusion prediction with system-call sequence-to-sequence model. IEEE Access 6:71413–71421. Michalenko, J. J.; Shah, A.; Verma, A.; Baraniuk, R. G.; Chaudhuri, S.; and Patel, A. B. 2019. Representing formal languages: A comparison between finite automata and recurrent neural networks. In Proc. ICLR 2019. OpenReview.net. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. Nature 518(7540):529–533. Okudono, T.; Waga, M.; Sekiyama, T.; and Hasuo, I. 2019. Weighted automata extraction from recurrent neural networks via regression on state spaces. CoRR abs/1904.02931. Omlin, C. W., and Giles, C. L. 1996. Extraction of rules from discrete-time recurrent neural networks. Neural Networks 9(1):41–52. Papernot, N.; McDaniel, P. D.; Swami, A.; and Harang, R. E. 2016. Crafting adversarial input sequences for recurrent neural networks. In Brand, J.; Valenti, M. C.; Akinpelu, A.; Doshi, B. T.; and Gorsic, B. L., eds., Proc. MILCOM 2016, 49–54. IEEE. Rabusseau, G.; Li, T.; and Precup, D. 2019. Connecting weighted automata and recurrent neural networks through spectral learning. In Chaudhuri, K., and Sugiyama, M., eds., Proc. AISTATS 2019, volume 89 of Proceedings of Machine Learning Research, 1630–1639. PMLR.