Decision-Guided Weighted Automata Extraction from Recurrent Neural Networks

Xiyue Zhang[1], Xiaoning Du[2*], Xiaofei Xie[2],
Lei Ma[3], Yang Liu[2], Meng Sun[1*]
[1] Peking University, China
[2] Nanyang Technological University, Singapore
[3] Kyushu University, Japan
{zhangxiyue, sunm}@pku.edu.cn, {xiaoning.du, xfxie, yangliu}@ntu.edu.sg, malei@ait.kyushu-u.ac.jp

Abstract

Recurrent Neural Networks (RNNs) have demonstrated their effectiveness in learning and processing sequential data (*e.g.*, speech and natural language). However, due to the black-box nature of neural networks, understanding the decision logic of RNNs is quite challenging. Some recent progress has been made to approximate the behavior of an RNN by weighted automata. They provide a better interpretability, but still suffer from poor scalability. In this paper, we propose a novel approach to extracting weighted automata with the guidance of a target RNN's decision and context information. In particular, we identify the patterns of its step-wise predictive decisions to instruct the formation of automata states. Further, we propose a state composition method to enhance the context-awareness of the extracted model. Our in-depth evaluations on typical RNN tasks, including language model and classification, demonstrate the effectiveness and advantage of our method over the state-of-the-arts. It achieves an accurate approximation of an RNN even on large-scale tasks.

1 Introduction

Over the past decade, recurrent neural networks (RNNs) have achieved great success in learning tasks of sequential data, such as natural language processing (Józefowicz et al. 2016; Mikolov et al. 2011) and speech recognition (Graves, Mohamed, and Hinton 2013; Zweig et al. 2017). To reduce impacts of vanishing gradient, the state-of-the-art RNNs (*e.g.*, LSTM, GRU) often adopt complex internal designs (*e.g.*, cell memory and gate control). However, the recursive computation and complex internal control design make the interpretation and understanding of RNNs rather challenging (Karpathy, Johnson, and Li 2015).

The stateful nature of RNNs inspired researchers to extract state transition rules through building an automaton to simulate the RNN behaviors. Up to present, existing attempts of stateful model extraction largely fall into two directions: *pedagogical approaches* and *compositional approaches* (Jacobsson 2005). Pedagogical approaches usually leverage exact learning algorithms, such as L* (Angluin 1987), to extract an exact behavior surrogate of an RNN. However, exact learning is mostly unable to scale up to practical RNN models with large alphabet size or states. On the other hand, the compositional approach first collects concrete state transitions of an RNN (by profiling with the training data). Then through applying state and transition abstraction, an automaton is built to approximate the behaviors of an RNN. This approach provides better scalability, and has been applied to adversarial attack and detection on industrial level tasks (Du et al. 2019; Dong et al. 2020). However, the compositional approach is often difficult to achieve a satisfactory approximation accuracy. In addition, the automaton extraction process is also unstable, highly depending on the abstraction configuration without systematic guidance.

To summarize, the major challenges along the automata extraction line are: (1) *Scalibility: to handle RNN tasks with high complexity*. Due to the exact-learning nature, techniques based on L* algorithm are usually limited to dealing with small synthetic tasks. However, RNNs used for practical applications are often of high complexity, which posts challenges to the scalability of the extraction methods. (2) *Precision: to generate automaton with accurate behavior approximation for the target RNN*. In the case of deterministic automata extraction through compositional approach, the extracted stateful model is usually with limited approximation accuracy. Besides, the lacking of guidance on hyper-parameter selection also impairs the exploration towards an optimal surrogate automaton. If the extracted automaton is inaccurate, *i.e.*, largely inconsistent with the RNN, behavior analysis w.r.t. the surrogate model can be unreliable to be applied to the original RNNs.

To this end, we propose a *decision-guided* approach for Weighted Finite Automaton (WFA) extraction from an RNN. Generally, our approach takes the state abstraction method aligned with the compositional approaches to deal with the scalability challenge, and leverages the weighted automaton to characterize the transition rules quantitatively for more precise extraction. To counteract the lacking of configuration guidance, we identify a finite set of decision patterns of the target model based on their probabilistic prediction tendency and utilize its size as a configuration estimation. Further, we integrate state composition to enhance the state-level context, which alleviates the impreciseness resulted from the contextual loss during the abstraction procedure. We also propose a synonym transition approach to

*Corresponding authors.

complementing the missing dynamics of new data, which realizes a significant improvement on precision towards natural language tasks. Overall, the proposed algorithm allows our approach to improve the extraction precision and scale up to complex application scenarios.

The main contributions are: (1) an algorithm for WFA extraction from RNN classifiers with better scalability and precision; (2) three important techniques including the configuration guidance for state abstraction by identifying decision patterns, the state composition to counteract contextual loss, and the synonym transition to circumvent missing dynamics of new data; (3) an implementation of the algorithm and evaluations over 15 benchmarks, including comparisons with the state-of-the-art (Weiss, Goldberg, and Yahav 2019).

2 Weighted Automaton Extraction Scheme

We first give the notations and concepts used in the paper, and then go through an RNN to WFA algorithm in Section 2.1. Solutions addressing the challenges are presented in Sections 2.2 to 2.4.

Notations Given a domain $\mathcal{D}$, $w \in \mathcal{D}^*$ denotes a sequence of elements from $\mathcal{D}$, and is of length $|w|$; ε denotes the empty sequence; x is used to denote an element in $\mathcal{D}$. $w[:i]$ indicates the prefix of w with the length i, w_i is the i-th element of w and $w \cdot x$ represents the sequence with x concatenated to w. $\mathbb{N}$ and $\mathbb{R}$ denote the sets of natural numbers and real numbers, respectively.

Definition 1 (RNN) *A Recurrent Neural Network (RNN) is a tuple $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{Y}, s_0, g, f)$, where $\mathcal{X}$ is the input space, $\mathcal{S}$ is the internal state space, $\mathcal{Y}$ is the probabilistic output space, $s_0 \in \mathcal{S}$ is the initial state, $g : \mathcal{S} \times \mathcal{X} \to \mathcal{S}$ is the transition function, and $f : \mathcal{S} \to \mathcal{Y}$ is the prediction function.*

Specifically, for an m-classification RNN with internal state of dimension n, the internal state space is a subset of $\mathbb{R}^n$ and the output space is a subset of $\mathbb{R}^m$, where $m, n \in \mathbb{N}$. The predictive outputs in the RNN classifier can either be logits or probabilities. In cases when the outputs are logits, we can apply the *softmax* function to the outputs to obtain the probabilistic outputs in the classification task.

RNN Configuration Here we look into the RNN configuration from two perspectives, and denote the configuration in the internal state space as $\delta_s(w)$ and the configuration in the probabilistic output space as $\delta_p(w)$. In particular, the configuration $\delta_s(w)$ is defined as $g^*(s_0, w)$, where $g^* : \mathcal{S} \times \mathcal{X}^* \to \mathcal{S}$ is the recursive application of g to a sequence. For $w \in \mathcal{X}^*$ and $x \in \mathcal{X}$, $g^*(s_0, w \cdot x) = g(g^*(s_0, w), x)$. The configuration $\delta_p(w)$ is defined as $f(g^*(s_0, w))$. Specifically, $\delta_s(\epsilon) = s_0$ and $\delta_p(\epsilon) = f(s_0)$.

To characterize the RNN behavior trace when dealing with an input sequence, we record its configuration sequentially after processing each input token, which forms a sequence of configurations. Given w of length n, the two types of configurations after taking the first i tokens are $\delta_s(w[:i])$ and $\delta_p(w[:i])$, with $0 \le i \le n$. We refer $[\delta_s(w[:i])]_{i=0}^n$ as the *internal state trace* of input w, and $[\delta_p(w[:i])]_{i=0}^n$ as the *decision trace* of input w.

The internal trace usually has a higher complexity and captures finer-grained behaviors of RNN than the decision trace. However, in the real world, the internal states are not always available either because the RNN is part of proprietary products or the possessor would like to keep it close-sourced as a black box.

Definition 2 (WFA) *A Weighted Finite Automaton over a finite alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, (E_\sigma)_{\sigma \in \Sigma}, I, T)$, where Q is a finite set of states; for any token $\sigma \in \Sigma$, E_σ is a transition matrix with size $|Q| \times |Q|$; I is the initial vector (a row vector), T is the final vector (a column vector), both of which are vectors of dimension $|Q|$.*

Given an input sequence w, its *weight* over WFA is computed through $I \cdot (\Pi_{i=1}^n E_{w_i}) \cdot T$. Starting from the initial vector, it multiplies the transition matrix corresponding to each input stimuli, and multiplies the final vector.

2.1 Weighted Automaton Extraction Algorithm

The steps to extract a WFA from a target RNN is elaborated in Alg. 1. It takes as inputs the target RNN $\mathcal{R}$, a set of input sequences X, and yields the extracted WFA $\mathcal{A}$. Intuitively, the WFA is established as a compact representation of RNN's behavior over the set of concrete inputs, and the quantitative weights enable it to be predictive on new inputs. As the first step (Lines 1-9), we execute $\mathcal{R}$ on the set of inputs X, and record the corresponding *internal state traces* as t (Line 5). For each input sequence w, we collect its stepwise transitions when seeing a new upcoming input element in the form of triples (Line 7), composed of the source state, triggering input element and the destination state. We also collect the configurations and the input tokens, preparing for the state abstraction and alphabet construction (Lines 8-9).

The next step is to develop an abstraction function for the collected RNN configurations (Line 10). Existing techniques usually leverage clustering algorithms to obtain the discrete partitions. However, they all face the problem of choosing an appropriate cluster number, the mis-selection of which can cause quite unstable and approximation too rough to be useful. Our proposed heuristic method for automatic cluster number estimation will be introduced later in Section 2.2. After obtaining the state abstraction function λ, we construct the abstract states of the weighted automaton and record the corresponding transition triples among the abstract states (Lines 14-17). For each token in the alphabet Σ, we follow the transition diagram construction approach in (Wang et al. 2018b) to collect and count the number of transitions that occurred between a state and its subsequent states. However, instead of only keeping the most frequent transitions, we introduce nondeterminism and calculate the transition matrix by applying normalization to the well-counted transition frequency among the states (Lines 18-24). The initial vector is the initial distribution over the abstract states, and here we set it to the one-point distribution (Line 25). Finally, we calculate the labeling vector uniquely for each state in Q (Lines 26-31). For a state q, we count the output labels corresponding to the instance RNN configurations mapped into q, and then transform the counts

Algorithm 1: Extraction of WFA from an RNN

input : $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{Y}, s_0, g, f)$: the target RNN,
X: input sequences
output: $\mathcal{A}$: the extracted WFA

$\Delta \leftarrow [\,]$; ▷ the array of transition triples
$D \leftarrow [s_0]$; ▷ the array of states
$\Sigma \leftarrow \emptyset$; ▷ the alphabet
for $w \in X$ **do**
  $t \leftarrow [\delta_s(w[:i])]_{i=0}^{|w|}$;
  for $i : 1 \mapsto |w|$ **do**
    $\Delta.add((t_{i-1}, w_{i-1}, t_i))$;
    $D.add(t_i)$;
    $\Sigma.add(w_{i-1})$;
$\lambda \leftarrow$ *fitStateAbstractor(D)* ; ▷ state abst. function
$\hat{\Delta} \leftarrow [\,]$; ▷ the array of abst. transition triples
$Q \leftarrow \emptyset$; ▷ the set of abst. states
$E \leftarrow \emptyset$; ▷ the set of transition matrices
for $(s, \sigma, s') \in \Delta$ **do**
  $\hat{\Delta}.add((\lambda(s), \sigma, \lambda(s')))$;
for $s \in D$ **do**
  $Q.add(\lambda(s))$;
for σ *in* Σ **do**
  for $q \in Q, q' \in Q$ **do**
    if $\hat{\Delta}.count((q, \sigma, \_)) > 0$ **then**
      $E_\sigma[q, q'] \leftarrow \frac{\hat{\Delta}.count((q,\sigma,q'))}{\hat{\Delta}.count((q,\sigma,\_))}$;
    else
      $E_\sigma[q, q'] \leftarrow \vec{0}$;
  $E.add(E_\sigma)$;
$q_0 \leftarrow \lambda(s_0), I \leftarrow \pi_{q_0}$;
for $q \in Q$ **do**
  $T[q] \leftarrow \vec{0}, labels \leftarrow []$;
  for $s \in \lambda^{-1}(q)$ **do**
    $labels \leftarrow$ *argmax*$(f(s))$;
  for $l \in set(labels)$ **do**
    $T[q, l] \leftarrow \frac{labels.count(l)}{|labels|}$;
return $\mathcal{A} = (Q, \Sigma, E, I, T)$

into a normalized label distribution. Note that λ^{-1} is the reverse function of λ, which returns the set of instance states mapped to a given abstract state.

Example We present the WFA extracted for a news classification task with our approach and show how it summarizes and interprets the knowledge learned by RNN. There are 7 labels in total, including *business*, *US*, *health* and four others. We assume an input sequence w = ["recipes", "for", "health", "roasted", "leeks"], whose label is *health*. Here we focus on the *key* token "health", and see whether RNN also deems it as an important implication for the *health* label. The extracted WFA contains 42 states. To ease the representation, we only keep the three states over which the probability distribution is mostly affected after taking "health". We assume the probability distribution over states before taking "health" is $I \cdot E_{recipes} \cdot E_{for}$=[0.89($q_0$), 0.03($q_1$), 0.0($q_2$)]. The transition matrix of token "health" is shown in Equation (1). The updated probability distribution after consuming "health" is [0.0(q_0), 0.54(q_1), 0.44(q_2)]. To better understand the influence of "health" over the prediction, we present the rows in the final matrix (formed by the final vector of each label) associated with the three states. For state q_0, q_1, and q_2, the top predictions are with probability $[0.47(US), 0.30(business)]$, $[0.40(US), 0.29(business), 0.25(health)]$ and $[0.94(health), 0.03(business)]$. We can see that q_2 is highly likely to yield label *health*. Intuitively, the transition matrix represents the specific semantics of "health" under this classification. q_2 is an absorbing state in the transition diagram of "health", and it is also of high probability for other states to transit to q_2. Interpreted in this way, we know that, basically, the semantics of "health" learned by RNN complies with human perception.

$$E_{health} = \begin{bmatrix} 0.0 & 0.56 & 0.42 \\ 0.0 & 0.17 & 0.83 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (1)$$

2.2 Decision-Guided State Abstraction

Challenge-I. As mentioned in Section 2.1, one of the big challenges for obtaining a superior state abstractor is the choice of cluster numbers. This is a common problem faced by clustering algorithms, *e.g.*, k-means.

To bridge the gap, we propose a method to estimate the cluster number by identifying the decision patterns. The probabilistic outputs stand out as a good choice for observing the patterns for two reasons. First, compared with the internal state layer, the probabilistic output layer is a summation from high-dimensional space to a low-dimensional space, which retains the important information and offers more computation feasibility in the meanwhile. Second, the probabilistic outputs carry clear semantics, which represent the prediction confidence on the inputs, thus can naturally inspire the heuristics of recognizing decision patterns. A larger probability predicted over a label indicates that the DL model is more confident about the decision.

For better illustration, we take an image classification task as an example. Fig. 1 displays three snapshots when an RNN consumes the pixels of an image row by row, and we observe the top-2 predicted classes ranked by probabilities. For the first snapshot, the classifier hesitates between "7" and "1" after processing the first third rows, and the confidence for both is quite similar. For the second snapshot, it tends to predict it as "0" after processing the first half information, but the top confidence (0.63) is still not that high. However, for the last one, after the model has a full view of the image, it recognizes the number "8" with high confidence (0.98). Interestingly, the model decisions observed from the probabilistic outputs highly comply with the human perception, as we also see both "0" and "9" from the middle image. With such inspiration, we try to capture the decision patterns from the perspective of predication confidence.

Technically, we import a configuration parameter t, and allow to split the probabilistic range $[0, 1]$ to t equal intervals and each probability value falls into one of the intervals. The resulted set of intervals is $\{[0, \frac{1}{t}), [\frac{1}{t}, \frac{2}{t}), \ldots, [\frac{t-1}{t}, 1]\}$ and

we denote it as I_t. We also define a function $\eta_t : y \mapsto I_t$ which maps a probability value to one of the t intervals.

Definition 3 (**k-DCP**) *In an m-classification task, given a probability distribution $(y_1, \ldots, y_m)$ with $\Sigma_{i=1}^{m} y_i = 1$, we define its k Decision Confidence Pattern (k-DCP) as $((c_1, \eta_t(y_{c_1})), \ldots, (c_k, \eta_t(y_{c_k}))$, where $(c_1, \ldots, c_k)$ are the ranked top-k predicted classes, and η_t is the interval mapping function with t as the splitting number.*

Now we introduce the concept of Decision Confidence Pattern (DCP), and define it in Definition 3. Note that another configurable parameter k is introduced to specify the number of primary prediction classes to observe when extracting the patterns. The configured DCP is called k decision confidence pattern, and is denoted as k-DCP. Intuitively, the k-DCP of a probability output captures the top-ranked class labels as well as their prediction confidence levels, *i.e.*, the probability intervals. Intuitively, if the outputs of two input samples derive the same k-DCP, the inputs may share similar features. Further, we define $dcp^{k,t} : \mathcal{Y} \mapsto ([1, m] \times I_t)^k$ as the function, which maps a probabilistic output to its k-DCP.

Finally, we demonstrate how to estimate the parameter (*i.e.*, the number of the clusters) for the clustering. Given an RNN $\mathcal{R}$ and an input sequence w, instead of focusing purely on the final output prediction, we take its decision trace, $[\delta_p(w[:i])]_{i=0}^{n}$, to better understand the gradual variation of decision patterns as with the consuming of input stimuli. Similarly, for a set of input sequences, we aggregate all the instance probabilistic outputs appearing in the decision traces to form a set of outputs Γ_p, formulated as $\Gamma_p = \{\delta_p(w[:i]) | w \in X, i \in [0, |w|]\}$ where X is the input sequence set. The set of decision patterns can be developed as $P = \{dcp^{k,t}(y) | y \in \Gamma_p\}$. Naturally, $|P|$ is the number of the decision confidence patterns we estimate for clustering algorithms (as in line 10 in Alg. 1). The parameters k and t could be configured to adjust the granularity of the patterns. In such a way, we obtain a "preview" on the variation diversity of the RNN prediction states, serving as an informative heuristic guidance for the abstraction degree of clustering.

Note that the estimated cluster numbers can be used as a guidance to construct the abstract states of automata either under the internal state space configuration or the probabilistic output configuration. We experimentally evaluated the performance variations when different configurations are utilized (cf. Section 3 RQ4).

2.3 Context-Aware State Composition

Challenge-II. The original RNN configurations (in the continuous domain) contain the exact contextual information for processing the next input token. However, with the abstraction, we lose a certain degree of the context precision as a trade-off for the generalization capability of the WFA. Precise contextual information is especially important for the accuracy of inference. Hence, we propose a further step to enhance the context characterization of WFA.

Inspired by n-grams algorithm, we propose a n-state composition approach, which synthesizes a state with its previous $n-1$ states and yields a new composite state, so as to enhance the contextual information. Given an input sequence w, and its internal state trace $[\tau_i]_{i=0}^{|w|}$, the i-th n-state is $\tilde{\tau_i} = (\tau_{i-n+1}, \ldots, \tau_i)$. Whenever there are less than $n-1$ states prior to a state, we pad with the first element of the sequence. Here, we define $\rho_n : \mathcal{D}^* \mapsto (\mathcal{D}^n)^*$ as the function to derive the n-state sequences from a sequence consisting of elements in domain $\mathcal{D}$. Then, the state abstraction can be applied to each component in the composite state, and derives the abstract n-states. The next steps to establish the WFA follows that in Alg. 1. Fig. 2 shows an example of state composition for 2-state. Via integrating the previous *one* state to each state in the top sequence, we obtain the composite 2-states in the bottom sequence.

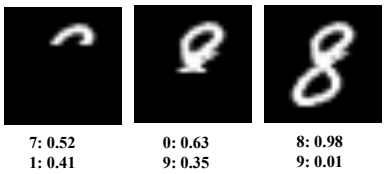

Figure 1: Examples of decision confidence.

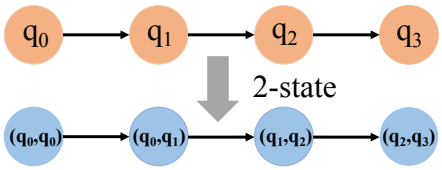

Figure 2: An example of 2-state composition.

2.4 Synonym Transition for New Data Tolerance

Challenge-III. When constructing the transition matrix of weighted automata (Lines 18 to 24 in Alg. 1), the matrix loses its integrity once there is no observed transitions from a source state via an input token. This is largely due to the fact that the construction is based on a finite set of data samples. However, in real practice, such as NLP tasks, it would be very common that previously unseen tokens appear in new input sequences. Thus, it is necessary to address this problem for a better tolerance of new data, and further circumvent missing dynamics in the inference phase.

Faced with the same problem, (Weiss, Goldberg, and Yahav 2019) suggested to use the uniform distribution over the next states for fairness, provided no information of transition dynamics were learned. However, this mitigation is only evaluated on small-scale formal language datasets and not on larger natural languages. Actually, filling in with uniform transitions would ignore the intrinsic semantics of never-seen tokens and lead to certain precision loss, which hinders the application on larger-scale real-world tasks where the amount of unknown tokens could be large.

For most RNN tasks, the (semantical) distance of input elements are measurable. Taking NLP tasks for example, according to our intuitive understanding on natural languages, the transition dynamics of two *synonyms* under the

Object			WL*			WFA			WFA_context			WFA (p)			WFA_context (p)		
Dataset	$\|\Sigma\|$	$loss$	CR	NDCG	Time(s)	CR	NDCG	Time(s)	CR	NDCG	Time(s)	CR	NDCG	Time(s)	CR	NDCG	Time(s)
SPiCe 0	4	1.16	0.87	0.97	95.4	0.88	0.96	1.6	0.93	0.97	1.9	0.85	0.95	1.6	0.94	0.97	5.2
SPiCe 1	20	2.77	0.86	0.97	142.5	0.95	0.70	51.5	0.96	0.70	140.4	0.61	0.67	47.7	0.76	0.68	482.7
SPiCe 2	10	2.13	0.90	0.96	477.4	0.95	0.75	84.9	0.96	0.75	133.3	0.57	0.71	50.1	0.88	0.74	520.6
SPiCe 3	10	2.14	0.57	0.88	373.9	0.82	0.85	96.5	0.86	0.85	258.9	0.76	0.81	92.8	0.81	0.82	387.0
SPiCe 4	33	1.74	0.55	0.63	324.1	0.69	0.82	1.2	0.77	0.77	99.2	0.70	0.80	1.2	0.81	0.73	184.6
SPiCe 6	60	1.68	0.41	0.57	1073.1	0.36	0.60	8.6	0.56	0.71	12.5	0.38	0.59	4.4	0.54	0.69	11.3
SPiCe 7	20	1.79	0.28	0.52	388.6	0.44	0.69	116.1	0.58	0.78	143.5	0.30	0.58	111.5	0.42	0.66	184.8
SPiCe 9	11	1.15	0.70	0.85	307.4	0.82	0.83	5.6	0.92	0.86	14.0	0.68	0.80	5.2	0.94	0.89	12.7
SPiCe 10	20	2.09	0.35	0.54	602.7	0.69	0.78	139.0	0.78	0.82	2481.4	0.66	0.75	128.7	0.79	0.82	3277.3
SPiCe 14	27	0.86	0.37	0.47	488.0	0.88	0.90	38.3	0.94	0.91	130.2	0.71	0.80	40.3	0.91	0.88	461.0
Average	22	1.75	0.59	0.73	427.3	0.75	0.79	54.3	0.83	0.81	341.5	0.62	0.75	48.3	0.78	0.79	552.7
UHL 1	2	0.70	1.00	1.00	18.0	1.00	0.99	2.3	1.00	0.99	2.6	1.00	1.00	2.3	1.00	0.99	2.6
UHL 2	5	1.32	1.00	1.00	99.5	1.00	0.96	29.6	1.00	0.97	47.1	1.00	0.96	28.9	1.00	0.98	35.9
UHL 3	2	0.84	0.76	0.99	72.9	0.81	0.96	20.8	0.82	0.96	26.1	0.84	0.99	20.4	0.99	0.98	38.9
Average	3	0.95	0.92	1.00	63.5	0.94	0.97	17.6	0.94	0.97	25.3	0.95	0.98	17.2	1.00	0.98	25.8

Table 1: Evaluation results on SPiCe and UHL datasets.

same context are expected to be similar as well. For instance, when processing two reviews, "this film is good" and "this film is great", inference on the final words "good" and "great" should trigger similar transition dynamics. With this insight, we propose a *synonym transition* method as the solution to fill in the blanks of transition dynamics for unseen tokens. When such tokens appear during the inference, we select its "synonym", characterized by the nearest distance on the embedding vectors, and apply the synonym's transition dynamics to the calculation. Specifically, to calculate the transition dynamics of an unseen token at a certain state , we first collect the tokens seen at that state and then rank them according to the distance with the newly-seen token. The corresponding row vector in the transition matrix of this synonym is then assigned to substitute the transition dynamics of the unseen token. Note that if the semantical distance between tokens is not measurable (*e.g.*, symbols in formal languages), we take the uniform distribution instead.

3 Experiments

This section is devoted to evaluating the effectiveness, scalability and usefulness of our approach. Four **Research Questions** (**RQs**) are to be answered: ❶ What is the approximation accuracy of the WFAs extracted through our approach? ❷ How effective is the context-aware state abstraction on improving the approximation accuracy? ❸ How effective is the synonym transition method, especially when applied to large-scale tasks? ❹ What is the performance of the WFA extracted from black-box RNNs?

Datasets and Baselines We selected WL* (Weiss, Goldberg, and Yahav 2019), the state-of-the-art technique for stateful model extraction from RNNs, as the baseline. For comparisons, we perform comprehensive evaluation with a total of 13 benchmarks, including 10 datasets from the SPiCe competition (Balle et al. 2017) and 3 artificial unbounded history languages (UHL) (2019). Besides, another two real-world datasets from NLP domain are further selected for evaluation of the scalability and usefulness, including the CogComp QC Dataset (abbrev. QC) (Li and Roth 2002) and the Jigsaw Toxic Comment Dataset (abbrev. Toxic) (Jigsaw 2018). We also tried WL* on these two datasets, but its scalability issue forces us unable to do so.

Experimental Settings All SPiCe and UHL datasets are split into training/validation/test sets with the percentage of 90%/5%/5% to train and test the RNN models. The RNN architectures follow the same configurations as in (Weiss, Goldberg, and Yahav 2019). For each dataset, a 2-layer LSTM network with 50 hidden dimensions is trained, with an exception for the SPiCe 4/6/9 datasets to be with 100 hidden dimensions and SPiCe 10/14 datasets with 20/30 hidden dimensions, respectively. For the QC dataset, we use 20K samples for training and 8K samples for testing, and train a single-layer LSTM with 32 hidden units, which achieves 83.0% test accuracy. For the Toxic dataset, we use 25k non-toxic samples and 25k toxic samples for model training (80%) and testing (20%). We train a single-layer LSTM model with 128 hidden units, which achieves 90.4% test accuracy. For all datasets, we established the automata based on the training datasets and evaluated its performance based on the test datasets.

Evaluation Metrics For our goal to extract an approximated model that is consistent with the target RNN's decision logic, two evaluation metrics, consistency rate (CR) and normalized discounted cumulative gain (NDCG), are used to evaluate the approximation precision of the extracted automata.

CR measures the ratio of consistency between the extracted automata and RNN prediction outputs over a dataset. For an input dataset $X = \{x_i\}_{i=1}^{N}$, the prediction results given by the automaton and the target RNN are $Y_a = \{y_{a_i}\}_{i=1}^{N}$ and $Y_r = \{y_{r_i}\}_{i=1}^{N}$, respectively. CR is calculated as follows:

$$CR(Y_a, Y_r) = \frac{\sum_{i \in N} \mathbb{I}(y_{a_i} = y_{r_i})}{N}$$

where $\mathbb{I}$ is an indicator function, which maps to 1 when $y_{a_i} = y_{r_i}$ and 0 otherwise.

NDCG measures the probabilistic prediction difference among the top k rankings. Given a m-classification task and an input w, assuming the automaton and RNN's top k prediction labels are $C_a = \{a_i\}_{i=1}^k$ and $C_r = \{r_i\}_{i=1}^k$, the probabilistic outputs of the RNN is $y_x = (y_i)_{i=1}^m$ then the NDCG is calculated as follows:

$$NDCG_k^x(C_a, C_r) = \frac{\sum_{i\in[1,k]} \frac{y_{a_i}}{\log_2(i+1)}}{\sum_{i\in[1,k]} \frac{y_{r_i}}{\log_2(i+1)}}$$

The NDCG score over a dataset X is calculated as the average score of all samples in X.

RQ1: Approximation Precision We conduct comparison experiments with WL* on the SPiCe and UHL datasets, in terms of CR, NDCG and extraction time cost (in seconds), and summarize the results in Table 1. The column "Object" shows the datasets, alphabet size and the test loss. The parameter k is set to 5 and 2, respectively, for calculating NDCG scores. The WL* approach is experimented with the default setting of variation tolerance, suffix and prefix thresholds in (Weiss, Goldberg, and Yahav 2019). For our approach, when setting the parameter k for DCP, we examine the cumulative prediction confidence of the top-k ranks, and select the k which makes its average on the training dataset reach 0.7. In this way, we focus the decision patterns on predictions dominating the model's confidence and achieve a good trade-off between precision and computation cost. The equipartion level t is set to 1 for the SPiCe datasets, and to $15/10/10$ for the UHL datasets. Basically, a finer-grained partition is recommended for tasks with lower-dimension probabilistic decision space.

Results on the three measurements are displayed in the "WL*" and "WFA" columns in Table 1 and our approach outperforms WL* on most of the datasets, especially on the complex ones. For SPiCe datasets, our approach improves the consistency rate and NDCG score with 16% and 6%, compared with that of WL*. In the meanwhile, the average extraction time cost of our approach is 54.3s, which leads to a reduction by 87.3% than WL*. Benefiting from the exact-learning feature, WL* demonstrates advantage w.r.t. NDCG score on the SPiCe 1 and SPiCe 2 datasets, which are relatively smaller synthetic languages, while our approach still outperforms with better CR scores. When the complexity of the tasks grows, either with more sophisticated prediction dynamics or with larger alphabet, our approach shows better approximation performance. For UHL datasets, our approach achieves better or comparable approximation accuracy on both metrics. As for the time cost, our approach takes several times less cost in all tasks. Moreover, the number of states in the extracted WFAs for the SPiCe and UHL datasets are 126 on average. Compared with WL*, the resulted automata realize a size reduction by 78.6%.

RQ2: Precision Enhancement by State Composition In this experiment, we investigated whether the explicit state composition for context enhancement could benefit the approximation performance. We performed experiments on SPiCe and UHL datasets under 2-state composition and reported the CR, NDCG metrics and the time cost under the column "WFA_context" in Table 1.

For SPiCe datasets, the application of state composition improves the consistency rate and NDCG score even further to 0.83/0.81, which is an obvious advantage compared to the 0.75/0.79 of the procedure without contextual enhancement. The average extraction time cost increases to 341.5s, but is still 20.1% lower than that of WL*. For UHL datasets, evaluations without the contextual enhancement already achieve an accurate approximation. After equipped with the state composition, our approach improves the consistency rate for UHL 3 and NDCG score for UHL 2 slightly, while achieving the same performance on other indicators.

Dataset	Acc. (%)	$\lvert\Sigma\rvert$	Config.	Synonym			Uniform		
				CR	NDCG	Time(s)	CR	NDCG	Time(s)
QC	83.0	17317	Internal	0.77	0.95	256.2	0.50	0.84	8.3
			Prob.	0.75	0.94	256.0	0.56	0.86	8.3
Toxic	90.4	24806	Internal	0.86	0.97	510.0	0.77	0.94	16.7
			Prob.	0.83	0.96	560.8	0.77	0.94	16.2

Table 2: Evaluation results of WFA extracted for QC and Toxic datasets based on internal/probabilistic space.

RQ3: Synonym Transition Effectiveness For the WFAs extraction of QC, k is set to 2 for the DCP recognition, following the same heuristics in RQ1, and the equipartition level t is set to 1. For the Toxic comment classification, the WFAs are extracted with the partition level t set to 50 and k of DCP set as 1 (with single-dimension logits returned by target models).

Table 2 shows the evaluation results on QC and Toxic classification tasks. The column "Acc." shows the test accuracy of the trained RNN models and the column "$|\Sigma|$" shows the alphabet size of the dataset. Column *CR*, *NDCG* and *Time* show the approximation precision in terms of consistency rate and NDCG score, and time cost of WFA extraction. We set the parameter of NDCG score as the label size of each dataset. The successful application of our approach on two real-world NLP classification tasks naturally demonstrates its scalability. Firstly, we look at the results under the "Synonym" column and with "Internal" configuration. For QC dataset, the constructed WFA based on internal configuration of the target RNN achieves 0.77 consistency rate and 0.95 NDCG score with extraction time as 256.2s. For Toxic dataset, the WFA extracted based on internal configuration makes a consistency rate of 0.86 and NDCG score of 0.97 while consuming time as 510.0s.

Now we look into the evaluations on how the synonym transition method counteracts the challenge of unknown transition dynamics when performing inference. In particular, we compare it with the uniform transition. The column "Uniform" in Table 2 displays the comparison results. The results confirm the advantage of synonym transition compared with the uniform method: the synonym transition method leads to better extraction precision across different RNN configurations and datasets. For instance, weighted automata extracted from the internal configurations under

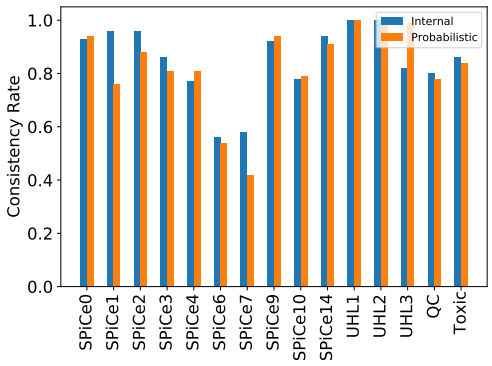

Figure 3: Consistency rate comparison on all datasets for WFAs extracted based on internal/probabilistic space.

synonym transition demonstrate an improvement on consistency rate by 54%/12% than that of uniform transition for QC/Toxic tasks. As a trade-off, the synonym transition method would lead to longer extraction time due to the distance calculation and order sorting among the tokens.

RQ4: WFA Extraction from Black-box RNNs We have shown that WFAs constructed based on target RNN's internal state space through our approach are able to approximate the target's behavior accurately. However, there are cases when the internal state space of a target model cannot be accessed. In such cases, we are only allowed to query the model and obtain the probabilistic outputs. In this experiment, we evaluate the effectiveness of our approach on modeling black-box RNNs. We perform experiments on the SPiCe, UHL, along with the QC and Toxic datasets, and extract the WFAs solely based on the probabilistic output space of the target RNNs, following the same configurations in evaluations on the internal state space.

The experimental results of SPiCe and UHL datasets are shown in columns "WFA(p)" and "WFA_context(p)" of Table 1 and the results of QC and Toxic datasets can be found in "Prob." rows of Table 2. Compared with WL*, the extracted WFAs under black-box setting still outperform WL* in almost all of the SPiCe tasks and achieve precise approximation on UHL datasets. Compared with the automata extracted from internal state space with contextual enhancement (column "WFA_context"), the approximation precision of automata extracted under black-box setting (column "WFA_context(p)") for SPiCe datasets decreases with 5% and 2% in terms of consistency rate and NDCG score on average. For UHL datasets, the WFAs constructed for black-box RNNs achieve accurate approximation and even slightly better than the internal-space extracted ones. For the large classification tasks, the performance of the constructed WFAs from black-box RNNs is discounted with 2%/3% regarding the consistency rate for QC/Toxic, and 1% regarding NDCG for both QC and Toxic tasks.

The bar chart in Fig. 3 shows the overall comparisons of consistency rate between the automata constructed from internal state space and probabilistic output space on all the datasets. Basically, automata extraction under these two types of configurations based on our approach demonstrate competitive performance. We further tested statistical significance of the difference in between with Mann-Whitney U test (Mann and Whitney 1947), and it is confirmed as non-significant with $p > 0.05$. We can conclude that WFAs extracted from black-box RNNs with our approach could still make a good approximation, which demonstrates the potential of our approach in assisting black-box applications (e.g. adversarial attacks).

4 Related work

Prior works have explored to extract finite state machines from RNNs. The earlier series of researches (Omlin and Giles 1996; Omlin, Giles, and Miller 1992; Omlin and Giles 1996; Schellhammer et al. 1998) focused on extracting a succinct and interpretable surrogate from RNN-acceptors (*i.e.*, Boolean-output RNNs). Jacobsson presented a review of research efforts in rule extraction (in the form of finite state machines) from RNNs and also highlighted that existing research mainly fall into two categories: pedagogical approaches and compositional approaches.

Along the pedagogical thread, a recent work (Weiss, Goldberg, and Yahav 2018) leveraged the exact learning algorithm L* to extract deterministic finite automaton (DFA) from RNN-acceptors. Later, they designed a weighted extension of L* algorithm to depict the behaviors of language model RNNs, with probabilistic deterministic finite automata. There are also works focusing on different types of RNNs other than classifiers, or limited to model a specific type of languages. Okudono et al. proposed a weighted extension of the L*-based procedure to extract weighted finite automata for real-value-output RNNs, and Ayache, Eyraud, and Goudian also focused on such RNNs but under a black-box setting. Their approaches are not applicable to RNN classifiers. Michalenko et al. investigated the relationship between RNN internal representations and finite automata for formal regular language recognition tasks. In contrast, our method supports automata extraction from black-box RNN classifiers, and can deal with both grammatical languages and natural languages.

Along the compositional thread, (Wang et al. 2018a,b) made attempts to examine key factors (*e.g.*, the grammar complexity, the clustering parameters) that may influence the reliability of extraction process of DFAs. Our approach encodes the RNN representation in a similar way, but leverages weighted automata, which captures the quantitative transition information, for better preciseness.

5 Conclusion

This paper proposed a decision-guided compositional approach to extract WFA from RNNs. We leveraged the diversity of decision confidence patterns to provide hints on the abstraction hyper-parameter selection. With the enhancement of contextual information, a finer-grained state abstraction were obtained with improved approximation precision. The design of the synonym transition also allowed better tolerance on new inputs. Effectiveness and scalibility of our approach were evaluated on two large-scale tasks in practice.

6 Acknowledgments

We thank the anonymous reviewers for their comprehensive feedback. This research was supported in part by the Guangdong Science and Technology Department (Grant No.2018B010107004); the National Natural Science Foundation of China under grant No.61772038, 61532019. It was also supported by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), the Singapore National Research Foundation under NCR Award Number NRF2018NCR-NSOE003-0001, NRF Investigatorship NRFI06-2020-0022; JSPS KAKENHI Grant No.20H04168, 19K24348, 19H04086, and JST-Mirai Program Grant No.JPMJMI18BB, Japan.

References

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75(2): 87–106. ISSN 0890-5401. URL https://doi.org/10.1016/0890-5401(87)90052-6.

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., *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, volume 93 of *Proceedings of Machine Learning Research*, 81–103. PMLR.

Balle, B.; Eyraud, R.; Luque, F. M.; Quattoni, A.; and Verwer, S. 2017. Results of the Sequence PredIction ChallengE (SPiCe): a Competition on Learning the Next Symbol in a Sequence. In Verwer, S.; van Zaanen, M.; and Smetsers, R., eds., *Proceedings of The 13th International Conference on Grammatical Inference*, volume 57 of *Proceedings of Machine Learning Research*, 132–136. Delft, The Netherlands: PMLR.

Dong, G.; Wang, J.; Sun, J.; Zhang, Y.; Wang, X.; Dai, T.; Dong, J. S.; and Wang, X. 2020. Towards Interpreting Recurrent Neural Networks through Probabilistic Abstraction.

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 *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, 477–487.

Graves, A.; Mohamed, A.; and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6645–6649.

Jacobsson, H. 2005. Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Comput.* 17(6): 1223–1263. ISSN 0899-7667. doi:10.1162/0899766053630350. URL https://doi.org/10.1162/0899766053630350.

Jigsaw. 2018. Toxic Comment Classification Challenge. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

Józefowicz, R.; Vinyals, O.; Schuster, M.; Shazeer, N.; and Wu, Y. 2016. Exploring the Limits of Language Modeling. *ArXiv* abs/1602.02410.

Karpathy, A.; Johnson, J.; and Li, F.-F. 2015. Visualizing and Understanding Recurrent Networks. *CoRR* abs/1506.02078.

Li, X.; and Roth, D. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, 1–7. Association for Computational Linguistics.

Mann, H. B.; and Whitney, D. R. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* 50–60.

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 *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Mikolov, T.; Kombrink, S.; Burget, L.; Černocký, J.; and Khudanpur, S. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5528–5531.

Okudono, T.; Waga, M.; Sekiyama, T.; and Hasuo, I. 2020. Weighted Automata Extraction from Recurrent Neural Networks via Regression on State Spaces. *AAAI* .

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

Omlin, C. W.; and Giles, C. L. 1996. Rule revision with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering* 8(1): 183–188.

Omlin, C. W.; Giles, C. L.; and Miller, C. B. 1992. Heuristics for the extraction of rules from discrete-time recurrent neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, 33–38 vol.1.

Schellhammer, I.; Diederich, J.; Towsey, M.; and Brugman, C. 1998. Knowledge Extraction and Recurrent Neural Networks: An Analysis of an Elman Network trained on a Natural Language Learning Task. In *New Methods in Language Processing and Computational Natural Language Learning*.

Wang, Q.; Zhang, K.; Ororbia, A. G.; Xing, X.; Liu, X.; and Giles, C. L. 2018a. A Comparative Study of Rule Extraction for Recurrent Neural Networks. *arXiv: Learning* .

Wang, Q.; Zhang, K.; Ororbia II, A. G.; Xing, X.; Liu, X.; and Giles, C. L. 2018b. An empirical evaluation of rule extraction from recurrent neural networks. *Neural computation* 30(9): 2568–2591.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2018. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine*

Learning, volume 80 of *Proceedings of Machine Learning Research*. Stockholmsmässan, Stockholm, Sweden: PMLR.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2019. Learning Deterministic Weighted Automata with Queries and Counterexamples. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8560–8571. Curran Associates, Inc.

Zweig, G.; Yu, C.; Droppo, J.; and Stolcke, A. 2017. Advances in all-neural speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4805–4809. IEEE.