Cold-Start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks

Chengyue Jiang[◇], **Yinggong Zhao**[†], **Shanbo Chu**[†], **Libin Shen**[†], **Kewei Tu**[◇*]
[◇] School of Information Science and Technology, ShanghaiTech University
Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Shanghai Engineering Research Center of Intelligent Vision and Imaging
[†] Leyan Technology. Inc
{jiangchy, tukw}@shanghaitech.edu.cn
{ygzhao, chushb, libin}@leyantech.com

This is a preprint version.
We will update and release the final version of the paper and code in a few weeks.

[*] Corresponding author.

Abstract

Neural networks can achieve impressive performance on many natural language processing applications, but they typically need large labeled data for training and are not easily interpretable. On the other hand, symbolic rules such as regular expressions are interpretable, require no training, and often achieve decent accuracy; but rules cannot benefit from labeled data when available and hence underperform neural networks in rich-resource scenarios. In this paper, we propose a novel type of recurrent neural networks called FA-RNNs that combine the advantages of neural networks and regular expression rules. An FA-RNN can be converted from regular expressions and deployed in zero-shot and cold-start scenarios. It can also utilize labeled data for training to achieve improved prediction accuracy. After training, an FA-RNN often remains interpretable and can be converted back into regular expressions. We apply FA-RNNs to text classification and observe that FA-RNNs significantly outperform previous neural approaches in both zero-shot and low-resource settings and remain very competitive in rich-resource settings.

1 Introduction

Over the past several years, neural network approaches have rapidly gained popularity in natural language processing (NLP) because of their impressive performance and flexible modeling capacity. Nevertheless, symbolic rules are still an indispensable tool in various industrial NLP applications. Regular expressions (RE) are one of the most representative and useful forms of symbolic rules and are widely used for solving tasks such as pattern matching and intent classification. RE-based systems are highly interpretable and therefore support fine-grained human inspection and manipulation. For example, individual RE rules in a system can be easily added, revised, or removed to quickly adapt the system to changes in the task specification. Moreover, RE-based systems do not require a training stage with labeled data and hence can be quickly deployed with decent performance in zero-shot scenarios. However, REs rely on human experts to write and often have high precision but moderate to low recall; RE-based systems cannot evolve by training on labeled data when available and thus usually underperform neural networks in rich-resource scenarios.

How to combine the advantages of symbolic rules and neural networks is an open question and is drawing increasing attention recently. One possible way is to use rules to constrain neural networks, usually in the manner of regularization via knowledge distillation (Hu et al., 2016) and multi-task learning (Awasthi et al., 2020; Xu et al., 2018), or by tuning the output logits of neural networks (Li and Srikumar, 2019; Luo et al., 2018). In this way, information from rules can be injected into neural networks, though the neural networks still require training and remain black boxes that are hard to interpret and manipulate. Another way of utilizing rules is to design novel neural network architectures inspired by rule systems (Schwartz et al., 2018; Graves et al., 2014; Peng et al., 2018; Lin et al., 2019). Models designed based on this idea usually achieve better interpretability, but they must be trained on labeled data and cannot be directly converted from rules or manually specified by human experts because of their structural differences from rule systems.

In this paper, we propose finite-automaton recurrent neural networks (FA-RNN), a novel type of recurrent neural networks that is designed based on the computation process of weighted finite-state automata. Because of the equivalence between REs and finite-state automata, we can convert any REs into an FA-RNN, which can be deployed in zero-shot and cold-start scenarios. When there are labeled data, the FA-RNN can also be trained in the same way as any neural network, which improves its prediction accuracy over the original REs. The FA-RNN has good interpretability. When converted from REs, it is (approximately) equivalent to the REs and is fully interpretable. Even after training, it often remains highly interpretable and can be converted back into REs. The interpretability of FA-RNNs opens the possibility of fine-grained manipulation such as integrating new REs into a trained FA-RNN and disabling old REs that are used to initialize an FA-RNN.

We apply FA-RNNs to the text classification task and compare them with neural network baselines as well as existing approaches of integrating REs and neural networks. Our experiments find that FA-RNNs show clear advantages in both zero-shot and low-resource settings and remain very competitive in rich-resource settings.

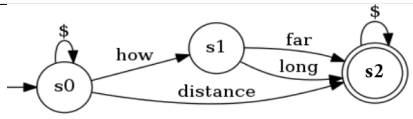
Label	[*distance*]
RE	*$*(how far | how long | distance) $**
Matched Text	⟨BOS⟩ tell me **how far** is oakland airport from downtown ⟨EOS⟩
FA	

Table 1: RE for matching sentences asking about distance, and a matched sentence. '$' is the wildcard. '|' is the **OR** operator. '*' is the Kleene star operator. We also show the finite automaton converted from the RE. s_2 is the final state.

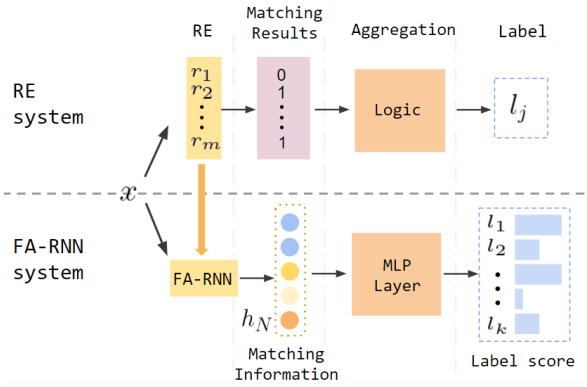


Figure 1: RE and FA-RNN systems for text classification.

2 Background

2.1 Regular Expressions

Regular expressions (RE) are patterns usually used for searching or matching a string and are a succinct way to denote regular languages. We show a simple example RE for matching sentences[1] in Table 1.

2.2 RE System for Text Classification

The text classification task aims to assign a class label to an input sentence. Let $\boldsymbol{x} = \langle x_0, \cdots, x_N \rangle$ be a sentence and $\mathcal{L} = \{l_1, \cdots, l_k\}$ be the label set. One common and straight-forward way to use REs for classification is as follows. Firstly, write m REs $\mathcal{R} = \{r_1, \cdots, r_m\}$, where each RE corresponds to some label in $\mathcal{L}$. Then, for each sentence $\boldsymbol{x}$, apply these REs to get matching results. Finally, aggregate the matching results to produce a final label for sentence $\boldsymbol{x}$ based on a set of propositional logic rules. Each rule specifies a logical expression of matching results that implies a specific label. For example, let M_i represent whether RE r_i is matched, then we may have a rule: $(M_i \vee M_j) \wedge \neg M_k \rightarrow l_p$. The whole procedure is shown in the top half of Figure.1.

2.3 Finite-State Automaton

Finite-state automata (FA) are machines with finite numbers of states. An FA can transit from one state to another in response to an input. It has one start state s_0 and a set of final states $\mathcal{S}_\infty$. Every RE can be converted into an FA expressing the same language by Thompson's construction algorithm (Thompson, 1968). For a sequence $\boldsymbol{x} = \langle x_1, \cdots, x_N \rangle$, an RE matches the sequence if and only if the converted FA starts from s_0 and finally reaches a final state after consuming $\boldsymbol{x}$. Table 1 shows an FA converted from the example RE. Further, for every RE, there exists a *unique* FA with a minimum number of states and deterministic transitions (m-DFA) such that they express the same language (Hopcroft et al., 2001). Deterministic transitions mean that given a current state and an input, there is a unique next state. The m-DFA can be obtained by running the powerset construction algorithm (Rabin and Scott, 1959) and the DFA minimization algorithm (Hopcroft, 1971).

2.4 Weighted Finite-State Automaton

A weighted finite-state automaton (WFA) assigns a weight to each transition, which is formally defined as a 5-tuple: $\mathcal{A} = \langle \Sigma, \mathcal{S}, \boldsymbol{T}, \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty \rangle$.

- Σ: a finite input vocabulary. $|\Sigma| = V$.
- $\mathcal{S}$: a finite set of states. $|\mathcal{S}| = K$.
- $\boldsymbol{T} \in \mathbb{R}^{V \times K \times K}$: a tensor of transition weights. $\boldsymbol{T}[\sigma, i, j]$ is the weight of transiting from s_i to s_j in response to input σ. $\boldsymbol{T}[\sigma] \in \mathbb{R}^{K \times K}$ denotes the transition matrix of σ.
- $\boldsymbol{\alpha}_0 \in \mathbb{R}^K$: initial weights. $\boldsymbol{\alpha}_0[i]$ is the weight of staying at state s_i at time $t = 0$.
- $\boldsymbol{\alpha}_\infty \in \mathbb{R}^K$: final weights. $\boldsymbol{\alpha}_\infty[i]$ is the weight of staying at state s_i after reading all the inputs.

An FA can be seen as a WFA with 0/1 weights. $\boldsymbol{T}[\sigma, i, j]$ is 1 if s_i can transit to s_j in response to σ and 0 otherwise. $\boldsymbol{\alpha}_0[i] = \mathbb{1}\{s_i = s_0\}$ and $\boldsymbol{\alpha}_\infty[i] = \mathbb{1}\{s_i \in \mathcal{S}_\infty\}$, where $\mathbb{1}()$ is the indicator function.

For sequence $\boldsymbol{x}$, the score of WFA $\mathcal{A}$ accepting $\boldsymbol{x}$ can be calculated using the forward (Baum and Petrie, 1966) and Viterbi (Viterbi, 1967) algorithms. Let path $\boldsymbol{p} = \langle u_1, \cdots, u_{N+1} \rangle$ be a sequence of indexes of the states that we visit when consuming $\boldsymbol{x}$. The score $\mathcal{B}(\mathcal{A}, \boldsymbol{p})$ of path $\boldsymbol{p}$ can be computed

[1] Example taken from the ATIS intent classification dataset.

by:

$$\boldsymbol{\alpha}_0[u_1] \cdot \left(\prod_{i=1}^{N} \boldsymbol{T}[x_i, u_i, u_{i+1}] \right) \cdot \boldsymbol{\alpha}_\infty[u_{N+1}] \quad (1)$$

Let $\pi(\boldsymbol{x})$ be the set of all paths that start from start state s_0 and reach a final state $s_i \in \mathcal{S}_\infty$ after consuming sequence $\boldsymbol{x}$. The forward algorithm computes the sum of path scores.

$$\begin{aligned} \mathcal{B}_{\text{forward}}(\mathcal{A}, \boldsymbol{x}) &= \sum_{\boldsymbol{p} \in \pi(\boldsymbol{x})} \mathcal{B}(\boldsymbol{\mathcal{A}}, \boldsymbol{p}) \\ &= \boldsymbol{\alpha}_0^T \cdot \left(\prod_{i=1}^{N} \boldsymbol{T}[x_i] \right) \cdot \boldsymbol{\alpha}_\infty \end{aligned} \quad (2)$$

The Viterbi algorithm computes the maximum of path scores.

$$\mathcal{B}_{\text{Viterbi}}(\mathcal{A}, \boldsymbol{x}) = \max_{\boldsymbol{p} \in \pi(\boldsymbol{x})} \mathcal{B}(\mathcal{A}, \boldsymbol{p}) \quad (3)$$

It can be computed by replacing matrix multiplication in Eqa.2 with the max-plus operator. For an FA $\mathcal{A}$, the forward score is exactly the number of paths in $\pi(\boldsymbol{x})$ while the Viterbi score indicates whether $\pi(\boldsymbol{x})$ is non-empty.

3 Method

We show step-by-step how we can convert REs to a novel type of recurrent neural networks called FA-RNNs.

3.1 From REs to Recurrent Neural Networks

RE to FA As mentioned in Sec.2.3, we can convert an RE into an m-DFA. In order to obtain a concise FA with better interpretability and faster computation speed, we treat the wildcard '$' as a special word in the vocabulary and run the algorithms mentioned in Sec.2.3 to obtain a "pseudo" m-DFA $\mathcal{A}$.

FA as RNN As discussed in Sec.2.4, the FA $\mathcal{A}$ can be seen as a WFA with 0/1 weights which is parameterized by $\Theta = \langle \boldsymbol{\alpha}_0, \boldsymbol{T}, \boldsymbol{\alpha}_\infty \rangle$.

The computation of the WFA forward score (Eqa.2) can be rewritten into a recurrent form. Let $\boldsymbol{h}_t \in \mathbb{R}^K$ be the forward score vector after consuming t words in $\boldsymbol{x}$. $\boldsymbol{h}_t[i]$ can be interpreted as the number of paths starting from s_0 and reaching s_i at step t.

$$\begin{aligned} \boldsymbol{h_0} &= \boldsymbol{\alpha}_0^T \\ \boldsymbol{h}_t &= \boldsymbol{h}_{t-1} \cdot \boldsymbol{T}[x_t],\ 1 \le t \le N \\ \mathcal{B}_{\text{forward}}(\mathcal{A}, \boldsymbol{x}) &= \boldsymbol{h}_N \cdot \boldsymbol{\alpha}_\infty \end{aligned} \quad (4)$$

The computation of the WFA Viterbi score can be formulated in a similar way. Therefore, we can view a WFA as a form of recurrent neural networks (RNN) parameterized by Θ.

3.2 Decomposing the Parameter Tensor

Despite the equivalence to FAs and hence better interpretability, the RNNs proposed in Sec.3.1 has much more parameters than a traditional RNN because of the tensor $\boldsymbol{T} \in \mathbb{R}^{V \times K \times K}$. To reduce the parameter number, we propose to apply tensor rank decomposition (explained in the supplementary material) and decompose $\boldsymbol{T}$ into three matrices $\boldsymbol{E}_\mathcal{R} \in \mathbb{R}^{V \times r}$, $\boldsymbol{D}_1 \in \mathbb{R}^{K \times r}$, $\boldsymbol{D}_2 \in \mathbb{R}^{K \times r}$, where r is a hyper-parameter. Note that if r is smaller than the rank of $\boldsymbol{T}$, then the decomposition is approximate. We empirically find that, for a 100-state FA converted from RE, we can obtain a small decomposition error ($\le 1\%$) if $r \ge 100$.

Now the RNN is parameterized by $\Theta_D = \langle \boldsymbol{\alpha_0}, \boldsymbol{\alpha_\infty}, \boldsymbol{E}_\mathcal{R}, \boldsymbol{D}_1, \boldsymbol{D}_2 \rangle$. $\boldsymbol{E}_\mathcal{R}$ has a dimension associated with vocabulary size V and can be viewed as a word embedding matrix containing RE information for each word. Let $\boldsymbol{v}_t \in \mathbb{R}^r$ be the embedding of word x_t contained in $\boldsymbol{E}_\mathcal{R}$. The recurrent update in Eqa.4 becomes:

$$\begin{aligned} \boldsymbol{a} &= (\boldsymbol{h}_{t-1} \cdot \boldsymbol{D}_1) \circ \boldsymbol{v}_t \\ \boldsymbol{h}_t &= \boldsymbol{a} \cdot \boldsymbol{D}_2^T \end{aligned} \quad (5)$$

where $\circ$ denotes element-wise product. Eqa.5 produces the same result as Eqa.4 with sufficiently large r.

Note that the size of $\boldsymbol{h_t}$ is determined by the state number K of the m-DFA converted from RE. In some cases, K may be too small, resulting in limited representational power of the RNN. A simple method to solve this problem is to concatenate $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ with a $K' \times r$ zero matrix, hence increasing the hidden state size by K'. Subsequent training (to be introduced later) would update $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ so that these added dimensions can be utilized. This is equivalent to adding K' isolated states in the FA and relying on training to establish transitions between the old and new states.

3.3 Integrating Pretrained Word Embedding

Pretrained word embeddings have been found very useful in bringing external lexical knowledge into neural networks. Let $\boldsymbol{E}_w \in \mathbb{R}^{V \times D}$ be the word embedding matrix and $\boldsymbol{u}_t \in \mathbb{R}^D$ be the word embedding of x_t in $\boldsymbol{E}_w$. We introduce another matrix

$\boldsymbol{G} \in \mathbb{R}^{D \times r}$ that can transform the D-dimensional word embedding $\boldsymbol{u}_t$ into r-dimension, which can then replace $\boldsymbol{v}_t$ in the recurrent update of Eqa.5. We initialize $\boldsymbol{G}$ by setting $\boldsymbol{G} = \boldsymbol{E}_w^{\dagger}\boldsymbol{E}_{\mathcal{R}}$, where $\boldsymbol{E}_w^{\dagger}$ is the pseudo-inverse of $\boldsymbol{E}_w$. In this way, we approximate $\boldsymbol{v}_t$ with $\boldsymbol{u}_t\boldsymbol{G}$ and hence the initialized RNN still tries to mimic the FA. After training, however, the RNN will be able to utilize the additional information contained in pretrained word embeddings and hence may outperform the original FA.

In practice, we find it beneficial to interpolate the two r-dimension embeddings $\boldsymbol{v}_t$ and $\boldsymbol{u}_t\boldsymbol{G}$ with a hyper-parameter $\beta \in [0, 1]$. When β is 1, we only use RE information. When β gets closer to 0, we integrate more external lexical information into the model. The recurrent update formula becomes:

$$\begin{aligned} \boldsymbol{z}_t &= \beta\boldsymbol{v}_t + (1-\beta)\boldsymbol{u}_t\boldsymbol{G} \\ \boldsymbol{a} &= (\boldsymbol{h_{t-1}} \cdot \boldsymbol{D}_1) \circ \boldsymbol{z}_t \\ \boldsymbol{h_t} &= \boldsymbol{a} \cdot \boldsymbol{D}_2^T \end{aligned} \tag{6}$$

We name this new form of RNNs as *FA-RNNs*, i.e., recurrent neural networks built from finite-state automata.

3.4 Extensions of FA-RNN

Gated Extension (FA-GRU) Inspired by the Gated Recurrent Unit (Chung et al., 2014), we sacrifice some interpretability and add an update gate $\boldsymbol{f}_t$ and a reset gate $\boldsymbol{r}_t$ into the FA-RNN. The update gate determines how much information from the past shall be retained. The reset gate determines whether to reset the previous score vector to $\boldsymbol{h}_1$, the score vector after reading ⟨BOS⟩ (Beginning of Sentence). The recurrent update is as follows.

$$\begin{aligned} \boldsymbol{z}_t &= \beta\boldsymbol{v}_t + (1-\beta)\boldsymbol{u}_t\boldsymbol{G} \\ \boldsymbol{f}_t &= \boldsymbol{\sigma}(\boldsymbol{W}_f\boldsymbol{z}_t + \boldsymbol{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \\ \boldsymbol{r}_t &= \boldsymbol{\sigma}(\boldsymbol{W}_r\boldsymbol{z}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r) \\ \hat{\boldsymbol{h}}_{t-1} &= (1-\boldsymbol{r}_t) \circ \boldsymbol{h_1} + \boldsymbol{r_t} \circ \boldsymbol{h_{t-1}} \\ \boldsymbol{a} &= (\hat{\boldsymbol{h}}_{\boldsymbol{t-1}} \cdot \boldsymbol{D}_1) \circ \boldsymbol{z}_t \\ \hat{\boldsymbol{h}}_t &= \boldsymbol{a} \cdot \boldsymbol{D}_2^T \\ \boldsymbol{h}_t &= (1-\boldsymbol{f}_t) \circ \boldsymbol{h}_{t-1} + \boldsymbol{f}_t \circ \hat{\boldsymbol{h}}_t \end{aligned} \tag{7}$$

$\boldsymbol{\sigma}$ is the sigmoid activation function and $\boldsymbol{W}_f$, $\boldsymbol{W}_r$, $\boldsymbol{U}_f$, $\boldsymbol{U}_r$ are additional parameters for gates. Note that when $\boldsymbol{f}_t$ and $\boldsymbol{r}_t$ is close to $\mathbf{1}$, the FA-GRU degenerates to the FA-RNN. Therefore, we initialize $\boldsymbol{b}_f, \boldsymbol{b}_r$ to a large value and $\boldsymbol{W}_f, \boldsymbol{W}_r, \boldsymbol{U}_f, \boldsymbol{U}_r$ randomly using Xavier initialization (Glorot and Bengio, 2010) to ensure that the initialized FA-GRU is approximately equivalent to the FA-RNN and hence the original REs.

Bidirectional Extension (BiFA-RNN) Our networks can be easily extended to their bidirectional variants. For any RE, we can reverse it by simply reversing its word order (e.g., "*free $* (phone | phones) $**" can be reversed to "*$* (phone | phones) $* free*") and then convert the reversed RE into WFA $\overleftarrow{\mathcal{A}}$ and the corresponding FA-RNN. Score vector $\overleftarrow{\boldsymbol{h_N}}$ can be computed by applying Eqa.6 or Eqa.7 on the reversed input sentence $\overleftarrow{\boldsymbol{x}}$. Then we take the average of $\overleftarrow{\boldsymbol{h_N}}$ and the left-to-right score vector $\overrightarrow{\boldsymbol{h_N}}$ to obtain the final score vector $\boldsymbol{h}_N = (\overrightarrow{\boldsymbol{h_N}} + \overleftarrow{\boldsymbol{h_N}})/2$.

3.5 Aggregation Layer for Text Classification

As introduced in Sec.2.2, an RE system for text classification contains multiple REs that are aggregated to form a class label prediction. Here we describe how to convert such an RE system to an FA-RNN system for text classification (the bottom half of Figure.1).

For each RE r_i in the RE system, we convert it into a WFA $\mathcal{A}_i$ with K_i states, start state $s_{0,i}$, and final weights $\boldsymbol{\alpha}_{\infty,i}$. We merge all the WFAs converted from the REs into a single WFA with $\sum_i K_i + 1$ states by adding a new universal start state s_0 connecting to all the individual state states $s_{0,i}$ via transition edges accepting ⟨BOS⟩ with weight 1. We then convert this WFA to an FA-RNN. After we run this FA-RNN on sentence $\boldsymbol{x}$, the last state vector $\boldsymbol{h}_N$ contains the matching information of all the REs.

To predict a class label from $\boldsymbol{h}_N$, we create a soft aggregation layer. First, we extract the forward or Viterbi score of each RE from $\boldsymbol{h}_N$. For forward scoring, we follow Eqa.4 and have:

$$\mathcal{B}_{\text{forward}}(\mathcal{A}_i, \boldsymbol{x}) = \boldsymbol{h}_N \cdot \bar{\boldsymbol{\alpha}}_{\infty,i} \tag{8}$$

where $\bar{\boldsymbol{\alpha}}_{\infty,i}$ expands $\boldsymbol{\alpha}_{\infty,i}$ by filling zeros for states not belonging to $\mathcal{A}_i$. For Viterbi scoring, we replace matrix multiplication with max-plus. The computed score for $\mathcal{A}_i$ can be seen as a soft matching result of RE r_i. Second, we rewrite the logical RE aggregation rules introduced in Sec.2.2 to soft logic expressions (Kimmig et al., 2012; Li and Srikumar, 2019) (Table 2). Instead of predicting a single label, the soft aggregation layer outputs the label logits $\boldsymbol{l} \in \mathbb{R}^k$. When all the elements in $\boldsymbol{h}_N$ are close to either 0 or 1, the output of the soft aggregation layer is approximately equivalent to that of the RE aggregation layer of Sec.2.2.

Logic	Soft Logic
$\neg A$	$1 - a$
$A \vee B$	$\min(1, a + b)$
$A \wedge B$	$\max(0, a + b - 1)$

Table 2: Soft logic. A, B are proposition symbols with soft truth values a, b.

Since the logical RE aggregation rules can be expressed in the conjunctive normal form, we can implement the corresponding soft aggregation layer with a two-layer MLP with ReLU-like activation functions. This is similar to the MLP layer commonly used at the end of traditional neural networks to map the hidden representation to label logits. In practice, we find it sometimes beneficial to not use any activation function in the MLP.

3.6 Training with Labeled Data

So far we have introduced how to initialize an FA-RNN system that is approximately equivalent to an RE classification system. When there are labeled data, the FA-RNN can also be trained to improve its performance. We simply use the output logits $\boldsymbol{l}$ to compute the cross-entropy loss on the training data and use a gradient-based method such as Adam (Kingma and Ba, 2014) to optimize it.

Note that we typically fix $\boldsymbol{E}_{\mathcal{R}}$ during training because we find that updating $\boldsymbol{E}_{\mathcal{R}}$ is not helpful. Therefore, the number of trainable parameters in an FA-RNN is similar to (usually smaller than) that of an RNN. We compare the number of parameters of different models in the supplementary material.

4 Experiments

We use the forward score version of FA-RNNs by default in our experiments. We use GloVe (Pennington et al., 2014) as the word embedding and keep it fixed for our methods and all the baselines. We tune the learning rate, number of additional isolated states K', and interpolation coefficient β for our methods on the development set. We provide more details of hyper-parameter tuning for FA-RNNs and all the baselines in the supplementary material.

4.1 Datasets

We evaluate the performance of our methods on three text classification datasets that have been used in previous work of integrating REs and neural networks: ATIS (Hemphill et al., 1990), Question Classification (QC) (Li and Roth, 2002) and SMS (Alberto et al., 2015). ATIS is a popular dataset consisting of queries about airline information and services. QC contains questions that can be classified into general categories like *LOCATION*, *ENTITY*, etc. SMS is a spam-classification dataset. We write REs for ATIS and use a modified version of REs from Awasthi et al. (2020) for QC and SMS. We show dataset statistics and RE examples in Table 3.

4.2 Baselines

Basic Networks We compare FA-RNN with traditional recurrent neural networks including RNN (Elman, 1990), GRU (Chung et al., 2014), LSTM (Hochreiter and Schmidhuber, 1997), and their bidirectional variants. We also experiment with a 4-layer CNN (Kim, 2014) and a 4-layer DAN (Iyyer et al., 2015), which are also frequently used in text classification. We feed the hidden representation produced by these models into an MLP to obtain the label logits and use the cross-entropy loss as the objective function. We tune the learning rates and the number of hidden states in [50, 100, 150, 200] on the development set for each dataset.

RE-enhanced Basic Networks We also compare our method with the basic neural networks enhanced by existing methods of combining rules and neural networks. Luo et al. (2018) propose three ways to utilize RE matching results in a neural model: 1) use the results as additional input features; 2) use the results to guide attention; 3) use the results to directly tune the output logits. As our basic networks do not involve attention, we enhance them using 1), 3) or both, denoted as *+i*, *+o* and *+io* respectively. Another method of utilizing rules is the knowledge distillation framework (Hinton et al., 2015). It treats the RE system as the teacher and its label logits as the soft targets, and distills this knowledge into the basic networks. We denote this method as *+kd*. Hu et al. (2016) combines knowledge distillation with posterior regularization by iteratively projecting the student network into the rule-regularized space. We denote this method as *+pr*.

4.3 Zero-Shot Classification

We compare our methods with the RE system and RE-enhanced BiGRU on the zero-shot scenario, in which no training data is available. All the methods use or are initialized by exactly the same set of REs.

	#Train	#Dev	#Test	$\|\mathcal{L}\|$	$\|\mathcal{R}\|$	K	%Acc
ATIS	3982	996	893	26	27	107	87.0
	*$ * flights | flight | ((go | get | fly) from $ * to $ *) $ * → FLIGHT*						
QC	4965	500	500	6	68	94	64.4
	*$ * what $? does $+ (stand? for) $* → ABBREVIATION*						
SMS	4502	500	500	2	36	52	93.2
	$ free $ * (phone | phones) $* → SPAM*						

Table 3: Dataset statistics and example REs. $\mathcal{L}$ is the label set. $\mathcal{R}$ is the RE set. K is the state number of the converted WFA. %Acc is the classification accuracy of the RE system. We provide an example RE and its targeting label for each dataset.

	ATIS	QC	SMS
FA-RNN	86.53	61.95	93.00
FA-GRU	86.81	**62.90**	**93.20**
BiFA-RNN	88.10	**62.90**	93.00
BiFA-GRU	**88.63**	**62.90**	**93.20**
BiGRU+*i*	2.66	17.25	71.15
BiGRU+*o*	44.68	37.35	50.75
BiGRU+*io*	27.94	37.05	68.30
BiGRU+*pr*	1.88	16.65	65.95
BiGRU+*kd*	1.88	16.65	65.95
RE system	87.01	64.40	93.20

Table 4: Accuracy of zero-shot classification. The RE system is included for reference.

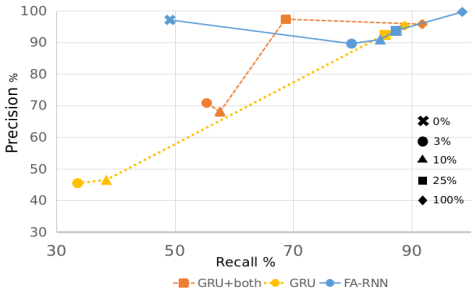


Figure 2: Precision and recall with different amounts of training data on SMS.

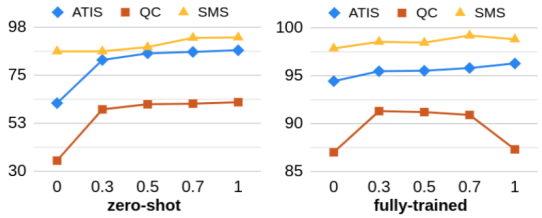


Figure 3: Performance of FA-RNN with different β.

We show the results in Table 4.

The results show that our methods have much better performance than RE-enhanced BiGRUs and are comparable to the RE system. RE-enhanced BiGRUs without training perform random guesses, except that in the cases of +*o* and +*io*, RE matching results directly influence the outputs and hence improve the predictions. The small differences in accuracy between the RE system and our methods are caused by approximation errors in decomposing the parameter tensor and integrating word embedding, as well as the introduction of gates in FA-GRUs.

We also report the results of the other baselines in the supplementary material. Without any training, the basic networks not enhanced by RE just perform random guesses. The other RE-enhanced basic networks have similar behaviors to RE-enhanced BiGRUs.

4.4 Low-Resource and Full Training

We compare all the methods trained on 1%, 10%, and 100% of the training data in Table 5. Because of space limit, for RE-enhanced networks, we only report the results of RE-enhanced BiGRUs, which perform the best among all the RE-enhanced networks. The complete results of all the methods with standard deviations can be found in the supplementary material.

From the results we can see that our methods outperform all the other methods in the low-resource settings, especially on 1% training data. With 100% training data, overall our methods are much better than RNN, DAN and CNN, and are either slightly better than or comparable to BiLSTM, BiGRU, and RE-enhanced BiGRUs. RE-enhanced BiGRUs are indeed better than non-enhanced BiGRU in general, and +*pr* and +*kd* seem to be more data-hungry than +*i*, +*o* and +*io*.

SMS is a binary classification task of spam detection, so we regard [*spam*] as the positive label and calculate the precisions and recalls of FA-RNN and two baselines GRU and GRU+*io* given different amounts of training data (Fig.2). With no training data, FA-RNN is almost equivalent to REs and has high precision but moderate recall; but with just 3% data, its recall is greatly improved and its precision drops only moderately; and with additional data, its precision and recall are both improved. For the baselines, the precision and recall of GRU always increase with more data, while the changes of the precision and recall of GRU+*io* seem less stable.

	ATIS (26-class)			QC (6-class)			SMS (2-class)		
	1%	10%	100%	1%	10%	100%	1%	10%	100%
FA-RNN	90.54	90.79	96.52	65.4	79.6	91.3	94.1	96.75	98.8
FA-GRU	90.73	90.85	96.61	66.65	80.7	91.85	94.15	96.8	99.2
BiFA-RNN	90.65	90.85	96.72	65.3	81.5	91.55	94.1	96.7	99
BiFA-GRU	89.89	90.26	96.64	66.15	82.8	91	94.15	96.75	98.8
CNN	76.71	86.09	94.74	46.65	74.9	89.25	89.35	95.9	98.8
DAN	74.5	83.68	90.4	49.45	65.4	77.8	88.75	93.7	98.6
RNN	71.19	75.17	91.55	39.1	67.9	85	89.8	89.85	97.75
LSTM	75.06	78.14	95.72	50.65	75.75	90	92.2	95.75	97.85
GRU	74.69	88.52	96.3	51.85	79.75	91.2	92.65	95.55	98.05
BiRNN	72.34	79.98	93.39	47	75.95	87.35	89.7	94.9	97.8
BiLSTM	75.87	87.12	96.25	47.35	76.75	90.95	92.15	95.8	97.7
BiGRU	76.12	88.35	96.75	48.35	80.05	91.5	92.4	95.95	98.4
BiGRU +*i*	82.42	90.01	96.56	64.3	80.25	92	93	96.75	98.55
BiGRU +*o*	77.35	89.22	96.33	52.15	80.2	91.7	93.05	95.95	98.4
BiGRU +*io*	82.61	89.95	95.46	59.6	79.65	90.7	93.15	96.75	98.25
BiGRU +*pr*	75.67	88.89	96.5	53.65	80.45	91.85	92.75	96.05	98.45
BiGRU +*kd*	75.5	88.86	96.75	49.2	80.3	91.25	92.15	96	98.55

Table 5: Classification accuracy with different amounts of training data.

FA-RNN	ATIS	QC	SMS
-F	**96.52**	**91.30**	98.80
-V	95.66	88.20	97.85
-F-O	94.51	87.80	**99.20**
-F-Rand	92.16	80.60	95.40
-V-Rand	91.26	78.60	97.00
-F-Rand$\boldsymbol{E}_w$	94.17	84.40	97.00
-Train$\boldsymbol{E}_{\mathcal{R}}$	96.41	89.20	99.00

Table 6: Ablation Study. -F denotes the default method using forward scoring. -V denotes Viterbi scoring. -O denotes the undecomposed version described in Sec.3.1. -Rand denotes random initialization. -Rand$\boldsymbol{E}_w$ denotes using random word embedding. -Train$\boldsymbol{E}_{\mathcal{R}}$ denotes training $\boldsymbol{E}_{\mathcal{R}}$.

5 Analysis

Impact of β β from Eqa.6 controls the influence of pretrained word embedding. Fig.3 shows how β impacts the performance of FA-RNN on the zero-shot and fully-trained scenarios. It can be seen that using pretrained word embedding does not help in the zero-shot scenario but can be helpful in the fully-trained scenario. One possible explanation is that word similarities encoded in the pretrained word embedding may not be compatible with a classification task, but training with data could adapt the model (in particular, by updating $\boldsymbol{G}$) to better utilize the information contained in the pretrained word embedding.

Ablation Study Table 6 shows the results of variants of FA-RNN when trained on the full datasets. From the results we can conclude that: 1) forward scoring outperforms Viterbi scoring; 2) tensor decomposition described in Sec.3.2 results in not only fewer parameters but also better overall performance; 3) RE initialization is helpful because it is much better than random initialization; 4) integrating pretrained word embedding is beneficial because the performance drops by a large margin if using random word embedding; 5) training $\boldsymbol{E}_{\mathcal{R}}$ does not result in better performance on average, possibly because it introduces too many trainable parameters.

6 Interpretability

An FA-RNN initialized with REs is approximately equivalent to the REs and therefore is interpretable. Even after training, an FA-RNN can still remain interpretable. We can use the trained parameters of the FA-RNN $\Theta_{\text{RE}} = \left\langle \hat{\boldsymbol{E}}_{\mathcal{R}}, \hat{\boldsymbol{D}}_1, \hat{\boldsymbol{D}}_2, \hat{\boldsymbol{G}} \right\rangle$ and word embedding matrix $\boldsymbol{E}_w$ to reconstruct the WFA tensor $\boldsymbol{T}$.

$$\begin{aligned} \hat{\boldsymbol{E}}_{w\mathcal{R}} &= \beta \cdot \hat{\boldsymbol{E}}_{\mathcal{R}} + (1-\beta) \cdot \boldsymbol{E}_w \hat{\boldsymbol{G}} \\ \hat{\boldsymbol{T}}_{(1)} &= (\hat{\boldsymbol{D}}_2 \odot \hat{\boldsymbol{D}}_1) \hat{\boldsymbol{E}}_{w\mathcal{R}}^T \end{aligned} \tag{9}$$

where $\hat{\boldsymbol{T}}_{(1)}$ denotes the mode-1 unfolding of the reconstructed tensor $\hat{\boldsymbol{T}}$ and $\odot$ denotes the Khatri-Rao product. Further, we can use a thresholding function $f(x) = \mathbb{1}\{x \geq \gamma\}$ to convert the weights into $\{0, 1\}$ to recover an FA, where γ is a fixed scalar. Similarly, we can round the weights in the soft aggregation layer to reconstruct the logical aggregation layer. In this way, we can convert a trained FA-RNN back into an RE system.

In our experiments, we find that although the reconstructed RE systems underperform the corresponding trained FA-RNNs because of thresholding and rounding during reconstruction, they often outperform the original REs. The reconstructed RE systems achieve 73.6% accuracy for QC ($+9.2\%$ compared with the original REs) and 87.45% for ATIS ($+0.45\%$ compared with the original REs). For SMS, the reconstructed REs underperform the original ones (-1.2%) probably because the original REs are already good enough. We show an example in Fig.4, in which our model can be seen to learn interesting new patterns such as 'jet' and '737'. We show another example in the supplementary material.

Good interpretability of our models opens the possibility of fine-grained manipulation of the

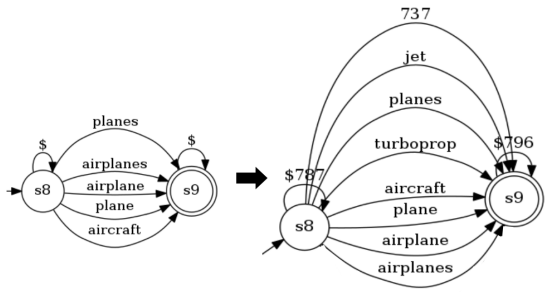


Figure 4: Part of an original RE and a reconstructed RE corresponding to label [*aircraft*]. On the right, '$787' means 787 words can activate the transition. Similar for '$796'. They are stricter than wildcard '$' that allows all possible words in the vocabulary.

model, e.g., adding new REs without retraining the model. To inject a new set of REs, we convert them to a new FA-RNN with parameters $\Theta_{\text{new}} = \langle \boldsymbol{E}_{\mathcal{R}}, \boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{G} \rangle$ and merge it into the original trained FA-RNN with parameters Θ_{RE} by concatenating the parameter matrices:

$$\left\langle [\hat{\boldsymbol{E}}_{\mathcal{R}}\ \boldsymbol{E}_{\mathcal{R}}], \begin{bmatrix} \hat{\boldsymbol{D}}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D}_1 \end{bmatrix}, \begin{bmatrix} \hat{\boldsymbol{D}}_2 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D}_2 \end{bmatrix}, [\hat{\boldsymbol{G}}\ \boldsymbol{G}] \right\rangle \quad (10)$$

To add new logical aggregation rules, we can update the aggregation layer parameters similarly by concatenation. To disable an RE in an FA-RNN, we reconstruct the WFA, remove all the states of the RE from the WFA except those that can be reached from states of other REs, and finally convert the WFA back to an FA-RNN.

7 Related Work

Neural Networks Enhanced by Rules Hu et al. (2016); Awasthi et al. (2020) use rules to constrain neural networks by knowledge distillation and multi-task learning. Rocktäschel et al. (2015); Xu et al. (2018); Hsu et al. (2018) use parsed rule results to regularize neural network predictions by additional loss terms. Li and Srikumar (2019); Luo et al. (2018) inject declarative knowledge in the form of parsed RE results or first-order expressions into neural networks by hacking the prediction logits or the attention scores. Hu et al. (2016); Hsu et al. (2018) use rules as additional input features. All these previous methods use matching results or truth values of rules to enhance existing neural models. In contrast, we directly turn REs into a novel type of trainable networks.

Relating Neural Networks and WFA Schwartz et al. (2018) propose a novel type of neural networks for learning soft surface patterns (a subset of REs), which is inspired by WFAs but cannot be converted from WFAs or surface patterns. In contrast, our FA-RNN can be initialized from REs and converted back to REs. Peng et al. (2018); Dodge et al. (2019) formulate the update of each hidden dimension of various RNN architectures as a small WFA (2-4 states). Weiss et al. (2018); Merrill (2019) provide theoretical analysis of various neural networks and their accepting languages. Our work differs from these more theoretical studies in that we aim for a practical text classification approach. Omlin et al. (1998); Giles et al. (1999) show the equivalence between WFA and second-order RNN. Our FA-RNN is first-order and does not use higher-order weights.

8 Conclusion and Future Work

We propose a novel type of recurrent neural networks called FA-RNN. It can be initialized from REs and can also learn from data, hence applicable to various scenarios including zero-shot, cold-start, low-resource and rich-resource scenarios. It is also interpretable and can be converted back into REs. Our experiments on text classification show that it outperforms previous neural approaches in both zero-shot and low-resource scenarios and is very competitive in rich-resource scenarios. In the future, we plan to apply FA-RNN to other tasks and explore other variants of FA-RNN. We release our data, RE rules and code at /path/to/code.

References

Túlio C Alberto, Johannes V Lochter, and Tiago A Almeida. 2015. Tubespam: Comment spam filtering on youtube. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 138–143. IEEE.

Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. 2020. Learning from rules generalizing labeled exemplars. *ICLR*.

Leonard E. Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Statist.*, 37(6):1554–1563.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Jesse Dodge, Roy Schwartz, Hao Peng, and Noah A. Smith. 2019. RNN architecture learning with sparse regularization. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1179–1184, Hong Kong, China. Association for Computational Linguistics.

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

C. L. Giles, C. W. Omlin, and K. K. Thornber. 1999. Equivalence in knowledge representation: automata, recurrent neural networks, and dynamical fuzzy systems. *Proceedings of the IEEE*, 87(9):1623–1640.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

John Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65.

Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. 2018. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 132–141, Melbourne, Australia. Association for Computational Linguistics.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. *ACL*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Angelika Kimmig, Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2012. A short introduction to probabilistic soft logic. In *NIPS 2012*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tao Li and Vivek Srikumar. 2019. Augmenting neural networks with first-order logic. *arXiv preprint arXiv:1906.06298*.

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

Chu-Cheng Lin, Hao Zhu, Matthew R Gormley, and Jason Eisner. 2019. Neural finite-state transducers: Beyond rational relations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 272–283.

Bingfeng Luo, Yansong Feng, Zheng Wang, Songfang Huang, Rui Yan, and Dongyan Zhao. 2018. Marrying up regular expressions with neural networks: A case study for spoken language understanding. *ACL*.

William Merrill. 2019. Sequential neural networks as automata. *CoRR*, abs/1906.01615.

Christian W Omlin, Karvel K Thornber, and C Lee Giles. 1998. Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks. *IEEE Transactions on Fuzzy Systems*, 6(1):76–89.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A Smith. 2018. Rational recurrences. *EMNLP*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Michael O Rabin and Dana Scott. 1959. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. 2015. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, Denver, Colorado. Association for Computational Linguistics.

Roy Schwartz, Sam Thomson, and Noah A Smith. 2018. Sopa: Bridging cnns, rnns, and weighted finite-state machines. *ACL*.

Ken Thompson. 1968. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision rnns for language recognition. *arXiv preprint arXiv:1805.04908*.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van Den Broeck. 2018. A semantic loss function for deep learning with symbolic knowledge. *35th International Conference on Machine Learning, ICML 2018*.