Neuralizing Regular Expressions for Slot Filling

Chengyue Jiang[◇], **Zijian Jin**[†], **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
[†] New York University
{jiangchy, tukw}@shanghaitech.edu.cn
zj2076@nyu.edu

Abstract

Neural models and symbolic rules such as regular expressions have their respective merits and weaknesses. In this paper, we study the integration of the two approaches for the slot filling task by converting regular expressions into neural networks. Specifically, we first convert regular expressions into a special form of finite-state transducers, then unfold its approximate inference algorithm as a bidirectional recurrent neural model that performs slot filling via sequence labeling. Experimental results show that our model has superior zero-shot and few-shot performance and stays competitive when there are sufficient training data.

1 Introduction

Neural approaches almost dominate recent natural language processing (NLP) research. The tremendous success of neural networks benefits from their strong capability of learning from a large amount of annotated data. On the other hand, systems based on symbolic rules, while being no longer the mainstream approach to NLP research, are still very widely used in practice because of their interpretability, trustworthiness and decent zero-shot performance in data-scarce scenarios. Therefore, how to integrate neural and symbolic approaches while retaining their respective advantages is becoming an active research direction.

In this paper, we try to integrate neural networks with regular expressions (RE), one of the most widely used forms of symbolic rules. We focus on the slot filling task (SF), a typical application scenario of REs that aims to identify words in a sentence that carry specific information. For example, given the sentence *"show me the flights from New York to Dallas."*, SF aims to tag the span *"New York"* as the content of slot *fr.city* (the departing city). An RE specifies text patterns of both contents and contexts and uses capturing groups to mark target contents.

Existing methods that integrate neural networks and symbolic rules either use rule outputs as pseudo-labels to distill knowledge into neural networks (Zhang et al., 2018; Hu et al., 2016), or use rule outputs to influence attention weights or output logits of neural networks (Luo et al., 2018; Wang et al., 2019; Li and Srikumar, 2019). These integration methods can outperform both rules and pure neural networks when there are sufficient training data. However, since they still require training, their performance is well below that of rules in zero-shot and low-resource settings. More recently, Jiang et al. (2020) proposed a novel method that converts REs into neural networks, which can match the performance of the original REs without training and can compete with previous integration methods when trained. Unfortunately, their method is designed for text classification and cannot be easily extended to the SF task because it does not model RE capturing groups and only computes sentence-level scores.

In this work, we propose to neuralize REs with capturing groups for the SF task. Our method is inspired by Jiang et al. (2020) but differs from their work in many aspects. Specifically, we treat SF as a sequence labeling task with the BIO tag scheme and convert REs with capturing groups into finite-state transducers (FST) with restricted output dependencies. We propose an approximate FST inference algorithm and unfold it as a neural model that resembles bidirectional RNN models for sequence labeling. Our model is approximately equivalent to the original REs and can be further improved by training on labeled data. We also propose several techniques to enhance the model without harming its initial approximation to the REs.

We conduct our experiments on three popular SF datasets involving two domains and two lan-

[*] Corresponding author.

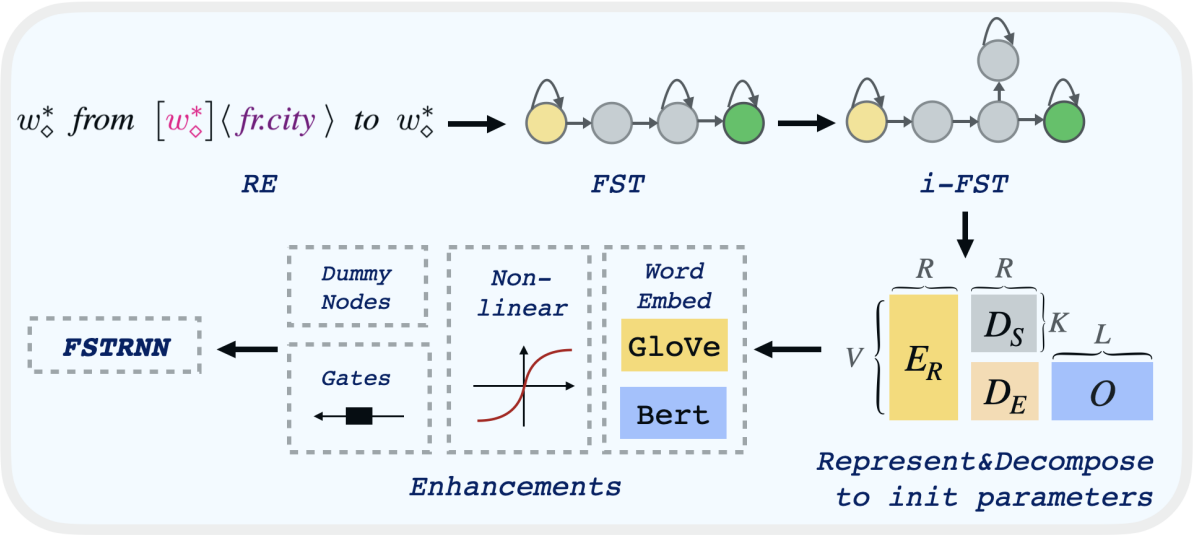


Figure 1: The procedure of neuralizing an RE system.

guages. Results show that our model has superior performance on zero-shot and low-resource settings compared with all the previous methods and remains competitive on rich-resource settings. We provide our source code as well as a handy toolkit for writing and converting REs in https://github.com/jeffchy/RE2NN-SEQ.

2 Background

2.1 Regular Expression for Slot Filling

We describe a slot filling system based on word-level REs with capturing groups. Consider the following RE that captures the content of a sentence for slot *fr.city* (the departing city).

$$w_\diamond^* \ \textit{from} \ \left[w_\diamond^*\right]\langle \textit{fr.city} \rangle \ \textit{to} \ w_\diamond^*$$

$w_\diamond$ is the wildcard symbol for words and it matches any word; symbol $*$ is the Kleene star operation which means the preceding symbol or sub-expression can appear for zero or more times. $\left[\text{sub-expression}\right]\langle$ *name* $\rangle$ is a capturing group that captures the contents matched by the sub-expression and tag them as *name*. An RE can have multiple capturing groups. To do slot filling, we write REs in which each capturing group is dedicated to a slot indicated by the *name*, and then we apply the REs one by one to capture the contents in the input sentences. For words captured by multiple groups with different names, we resolve the conflicts based on pre-defined priority. For example, given sentence *show me the flights from New York to Dallas*, the RE above would identify "*New York*" for slot *fr.city*.

2.2 Finite-State Transducer

A finite-state transducer (FST) is a finite-state machine with input and output. At each time step, it accepts a word, transits from one state to another, and outputs a label. It is formally defined as a 6-tuple: $\mathcal{T} = \langle\, Q, \Sigma, \Gamma, Q_I, Q_F, \Omega \,\rangle$.

- Q: a finite set of states. $|Q| = K$.
- Σ: a finite input vocabulary. $|\Sigma| = V$.
- Γ: a finite output vocabulary. $|\Gamma| = L$.
- Q_I: a finite set of start states, a subset of Q.
- Q_F: a finite set of final states, a subset of Q.
- $\Omega = \Sigma \times \Gamma \times Q \times Q$: all possible transitions. $\times$ denotes cartesian product.

If we assign a weight to each transition and a weight to each state for being a start state or final state, then we get a weighted finite-state transducer (WFST). We can use a 4th-order tensor $\boldsymbol{T}_\Omega \in \mathbb{R}^{V \times L \times K \times K}$ to represent the transition weights, and two K-dimension vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ to represent the weights of start and final states. An FST can be viewed as a WFST with 0/1 weights. Specifically, $\boldsymbol{T}_\Omega[m, n, i, j] = 1$ iff. $\mathcal{T}$ accepts input token σ_m, transits from state q_i to state q_j, and outputs γ_n, where $\sigma_m \in \Sigma, \gamma_n \in \Gamma, q_i, q_j \in Q$. $\boldsymbol{\mu}[i] = 1$ iff. $q_i \in Q_I$ and $\boldsymbol{\nu}[j] = 1$ iff. $q_j \in Q_F$.

We define a path as a sequence of transitions. An FST **accepts** an input sequence if there exists at least one path that starts from a start state, matches the input sequence at each position, and ends at a final state. We call such a path an **accepting path**, which defines a mapping from the input sequence to an output sequence. The score of path $\omega_1, \cdots, \omega_m$ with start state q_i and final state q_j can

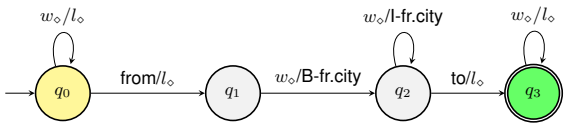


Figure 2: An example finite state transducer. q_0 is the only start state and q_3 is the only final state. $w_\diamond$ is the wildcard for input words and $l_\diamond$ is the wildcard for output labels. Each arc represents a possible transition and the slash above each arc separates the input (left) and the output (right).

be computed as: $\boldsymbol{\mu}[i] \cdot (\prod_{k=1}^{m} \boldsymbol{T}_\Omega(\omega_k)) \cdot \boldsymbol{\nu}[j]$, where $\boldsymbol{T}_\Omega(\omega_k)$ denotes the weight of transition ω_k.

We show an example FST in Fig. 2. The input *"flights from New York to Dallas"* can be accepted by the FST, and the accepting path indicates a state sequence $[q_0, q_0, q_1, q_2, q_2, q_3, q_3]$ and output sequence $[l_\diamond, l_\diamond,$ *B-fr.city*, *I-fr.city*, $l_\diamond, l_\diamond]$.

3 Method

In this section, we introduce the several steps of neuralizing REs (as shown in Fig. 1) as well as the decoding and training methods.

3.1 Converting RE to FST

Sakuma et al. (2012) rigorously show that any RE with capturing groups can be converted into an FST. The equivalence of the RE and FST is reflected in: (1) RE matches a sentence if and only if the corresponding FST has at least one accepting path; (2) the output sequence of the FST matches the content captured by the RE. Here we use the BIO tag scheme in the FST output to specify captured groups. For example, we tag *New* and *York* as *B-fr.city* and *I-fr.city* to represent the captured group *New York* as *fr.city*. For words outside of any capture group (e.g., '*from*', '*Dallas*' in the example sentence (Sec. 2.1)), however, we depart from the BIO scheme and assign a wildcard label $l_\diamond$ instead of the outside label '*O*' to them, which means the RE is totally unsure about which label the words should be assigned because groups in other REs might capture them. For example, '*Dallas*' can be captured by another RE as '*to.city*' (arriving city) and hence shall not be assigned label '*O*'.

To convert an RE to an FST, we view an FST as a finite-state automaton with vocabulary $\Sigma \times \Gamma$, use Thompson's construction (Thompson, 1968) to build the automata from the RE and further minimize the automata using Hopcroft's algorithm (Hopcroft, 1971). In the resulting FST, the input vocabulary Σ is the RE vocabulary and the output vocabulary Γ contains labels *B-X* and *I-X* for each slot X, as well as O and the wildcard label $l_\diamond$. As an example, the RE in Sec. 2.1 can be converted into an FST shown in Fig. 2. We provide the complete conversion algorithm and prove its correctness in Appendix A. When there are multiple REs, we take the union of the REs with the 'or' operation (i.e., a|b) to form a big RE and turn it into an FST.

Algorithm 1: Inference in FST

Input: $\boldsymbol{x} = x_1, \cdots, x_m$. $\mathcal{T} = \langle \boldsymbol{T_\Omega}, \boldsymbol{\mu}, \boldsymbol{\nu} \rangle$
Step 1: sum out the dimension w.r.t. label for tensor $\boldsymbol{T}_\Omega$ to get $\boldsymbol{T}'_\Omega \in \mathbb{R}^{V \times K \times K}$. Let $\boldsymbol{T}'_\Omega[x_t]$ denote the transition matrix of word x_t from $\boldsymbol{T}'_\Omega$
Step 2: calculate forward scores. let $\boldsymbol{\alpha}_0 = \boldsymbol{\mu}^T$.
for $t \leftarrow 1$ **to** m **do**
  $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} \cdot \boldsymbol{T}'_\Omega[x_t]$.
Step 3: calculate backward scores. let $\boldsymbol{\beta}_m = \boldsymbol{\nu}^T$.
for $t \leftarrow m$ **to** 1 **do**
  $\boldsymbol{\beta}_{t-1} = \boldsymbol{\beta}_t \cdot \boldsymbol{T}'_\Omega[x_t]^T$.
Step 4: get label scores $\boldsymbol{c}_t \in \mathbb{R}^L$ at each position t
$(\boldsymbol{c}_t)_k = (\boldsymbol{\alpha_{t-1}})_i \, (\boldsymbol{T}_\Omega[x_t])_{kij} \, (\boldsymbol{\beta_t})_j$ (einsum notation)

3.2 Inference in FST

Given sentence $\boldsymbol{x} = x_1, \cdots, x_m$, FST inference aims to find the output label sequence $\boldsymbol{y} = y_1, \cdots, y_m$. The score of $\boldsymbol{y}$ is defined as the weight sum of all the accepting paths matching input $\boldsymbol{x}$ and output $\boldsymbol{y}$. As finding the highest scoring output given a sentence is proved to be NP-hard (Casacuberta and De La Higuera, 2000), we instead use an approximate inference that finds the highest scoring output label at each position independently of the labels at all the other positions. This can be done with the classic variable elimination algorithm, which involves a forward process summing out variables to the left and a backward process summing out variables to the right of each position. We show the algorithm of computing label scores simultaneously at all the positions in Algorithm 1.

3.3 Independent FST

The 4th-order tensor in the FST leads to the following issues. (1) High space complexity $\mathrm{O}(VLK^2)$. (2) High time complexity for each position: $\mathrm{O}(LK^2)$. (3) Hard to decompose the tensor (see Sec. 3.4).

FST to i-FST We address these concerns by transforming the FST such that each label output is independent of the input and the source state

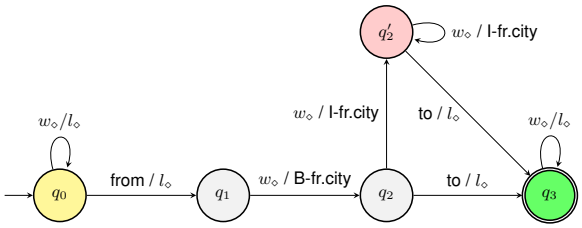


Figure 3: An example i-FST. q_2' is the added state.

Algorithm 2: Inference in i-FST.

1 **Input:** $\boldsymbol{x} = x_1, \cdots, x_m$. $\mathcal{T} = \langle \boldsymbol{T}, \boldsymbol{O}, \boldsymbol{\mu}, \boldsymbol{\nu} \rangle$
2 **Step 1:** sum out the dimension w.r.t. label for $\boldsymbol{O}$ to get vector $\boldsymbol{o} \in \mathbb{R}^K$. Let $\circ$ denote element-wise multiplication.
3 **Step 2:** calculate forward scores, let $\boldsymbol{\alpha}_0 = \boldsymbol{\mu}^T$.
4 **for** $t \leftarrow 1$ **to** m **do**
5 $\quad \boldsymbol{\alpha}_t = (\boldsymbol{\alpha}_{t-1} \cdot \boldsymbol{T}[x_t]) \circ \boldsymbol{o}^T$.
6 **Step 3:** calculate backward scores, let $\boldsymbol{\beta}_m = \boldsymbol{\nu}^T$.
7 **for** $t \leftarrow m$ **to** 1 **do**
8 $\quad \boldsymbol{\beta}_{t-1} = (\boldsymbol{\beta}_t \circ \boldsymbol{o}^T) \cdot \boldsymbol{T}[x_t]^T$.
9 **Step 4:** get label scores $\boldsymbol{c}_t \in \mathbb{R}^L$ at each position t.
10 $\boldsymbol{c}_t = (\boldsymbol{\alpha}_t \circ \boldsymbol{\beta}_t) \cdot \boldsymbol{O}$

of the transition given the target state. We call it independent FSTs (i-FST). Fig. 3 shows an i-FST converted from the FST shown in Fig. 2. In the example i-FST, the output is *B-fr.city* if we reach q_2, but in the original FST, the output could be either *B-fr.city* or *I-fr.city* depending on the source state. We show the conversion algorithm in Appendix B. The algorithm runs in polynomial time $O(LK^3)$ and adds at most $O(LK)$ new states. As FSTs converted from REs naturally satisfy this independence condition in most cases, the algorithm runs much faster and adds no more than 50% new states in our experiments on three datasets.

3th-order representation Because of the independence in an i-FST, we can use a 3th-order tensor $\boldsymbol{T} \in \mathbb{R}^{V \times K \times K}$ to represent transitions between states given an input, and a matrix to represent the mapping from end states to the output label $\boldsymbol{O} \in \mathbb{R}^{K \times L}$. We can use $\boldsymbol{T}$ and $\boldsymbol{O}$ to recover the 4th-order tensor of an i-FST: $\boldsymbol{T}_\Omega[\sigma, \gamma, i, j] = \boldsymbol{T}[\sigma, i, j] \times \boldsymbol{O}[j, \gamma]$.

Inference in i-FST We can again apply variable elimination to compute label scores (Algorithm 2). Since the output label now depends only on one state, the time complexity is also reduced. We compare FST and i-FST in Table 1.

We find that i-FST inference resembles running a *BiRNN* model with a linear output layer for sequence labeling. We recurrently update the forward score $\boldsymbol{\alpha}_t$ and backward score $\boldsymbol{\beta}_t$ based on the input x_t and the previous scores at each position, so $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ resemble the forward and backward hidden states in a *BiRNN*. At position t, we aggregate $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ and map them to the label score vector y_t, which also resembles concatenating the bidirectional hidden states together and feeding them into the output layer in a *BiRNN*.

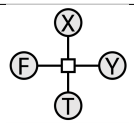
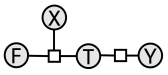
	FST	i-FST
space	$O(VLK^2)$	$O(VK_1^2)$
time at each position	$O(LK^2)$	$O(LK_1 + 2K_1^2)$
scoring	transition	reaching state
factor graph		

Table 1: Comparison of FST and i-FST. K_1 denotes the number of states for i-FST. In practice, $K_1 \leq 1.5K$. In the factor graphs, variable **S** and **E** represents the source and target states of a transition, and **X**, **Y** represent the input and output.

3.4 Parameter Tensor Decomposition: the last step towards *FSTRNN*

An i-FST is much more compact and faster but it still: (1) has too many parameters (especially $\boldsymbol{T} \in \mathbb{R}^{V \times K \times K}$) compared to a traditional *BiRNN*; and (2) is unable to incorporate external word embeddings. To tackle these problems, we adopt the tensor decomposition-based method proposed by Jiang et al. (2020) and modify the forward and backward score computation accordingly (Steps 3 and 4 in Algorithm 2).

CP Decomposition (CPD) We apply CPD to decompose the 3th-order tensor $\boldsymbol{T}$ into three matrices: $\boldsymbol{E}_\mathcal{R} \in \mathbb{R}^{V \times R}, \boldsymbol{D}_S \in \mathbb{R}^{K \times R}, \boldsymbol{D}_E \in \mathbb{R}^{K \times R}$, where R is rank, a hyper-parameter. A higher rank usually results in lower decomposition errors. We show the detail of CPD in Appendix C. After tensor decomposition, we use $\boldsymbol{E}_\mathcal{R}, \boldsymbol{D}_S, \boldsymbol{D}_E$ instead of $\boldsymbol{T}$ to compute forward and backward scores. Line 5 of Algorithm 2 becomes:

$$\begin{aligned} \boldsymbol{v}_t &= \boldsymbol{E}_\mathcal{R}[x_t] \\ \boldsymbol{g} &= (\boldsymbol{\alpha}_{t-1} \cdot \boldsymbol{D}_S) \circ \boldsymbol{v}_t \\ \boldsymbol{\alpha}_t &= (\boldsymbol{g} \cdot \boldsymbol{D}_E^T) \circ \boldsymbol{o}^T \end{aligned} \tag{1}$$

where $\boldsymbol{E}_\mathcal{R} \in \mathbb{R}^{V \times R}$ can be treated as a special word embedding matrix derived from the original

RE, and $\boldsymbol{v}_t$ is the embedding of word x_t selected from $\boldsymbol{E}_{\mathcal{R}}$. If $\boldsymbol{E}_{\mathcal{R}}, \boldsymbol{D}_S, \boldsymbol{D}_E$ reconstruct $\boldsymbol{T}$ perfectly, then Eqa. 1 is equivalent to line 5 in Algorithm 2. Similarly, the backward score equation in line 8 is modified to:

$$\begin{aligned} \boldsymbol{v}_t &= \boldsymbol{E}_{\mathcal{R}}[x_t] \\ \boldsymbol{a} &= ((\boldsymbol{\beta}_t \circ \boldsymbol{o}^T) \cdot \boldsymbol{D}_E) \circ \boldsymbol{v}_t \\ \boldsymbol{\beta}_{t-1} &= (\boldsymbol{a} \cdot \boldsymbol{D}_S^T) \end{aligned} \tag{2}$$

Incorporating External Word Embedding As mentioned above, we treat $\boldsymbol{E}_{\mathcal{R}}[x_t]$ as an R-dimension word vector. We may also want to inject additional information of the word by incorporating externally pretrained word embedding.

For static word embeddings such as GloVe, we again adopt the method of Jiang et al. (2020). Let $\boldsymbol{E}_w \in \mathbb{R}^{V \times D}$ denote the word embedding matrix we want to incorporate. $\boldsymbol{E}_w[x_t]$ is the D dimension word vector for x_t. To combine these two word vectors together, we introduce a $D \times R$ trainable matrix $\boldsymbol{G}$ that maps the pretrained word vector from D dimension to R dimension. $\boldsymbol{G}$ is initialized as $\boldsymbol{E}_w^{\dagger}\boldsymbol{E}_{\mathcal{R}}$, where $\boldsymbol{E}_w^{\dagger}$ is the pseudo-inverse of $\boldsymbol{E}_w$. As a result, $\boldsymbol{E}_w[x_t] \cdot \boldsymbol{G}$ approximates $\boldsymbol{E}_{\mathcal{R}}[x_t]$. We then interpolate these two R-dimension vectors with hyper-parameter η to get new $\boldsymbol{v}_t$, and use it to calculate the forward and backward scores in Eqa. 1 and 2.

$$\boldsymbol{v}_t = \eta \boldsymbol{E}_{\mathcal{R}}[x_t] + (1-\eta)\boldsymbol{E}_w[x_t] \cdot \boldsymbol{G} \tag{3}$$

To incorporate contextualized word embeddings such as BERT, we first use two methods to acquire an intermediate static word embedding matrix $\boldsymbol{E}_w \in \mathbb{R}^{V \times D}$: (1) The **Aggregate** method proposed by Bommasani et al. (2020); (2) A **Random** matrix $\boldsymbol{E}_w \sim \mathcal{N}(0, 0.5)$. Then we initialize $\boldsymbol{G}$ as $\boldsymbol{E}_w^{\dagger}\boldsymbol{E}_{\mathcal{R}}$. The new word embedding $\boldsymbol{v}_t$ for x_t can be obtained by Eqa.4, where $\boldsymbol{u}_t$ is the contextual word embedding of x_t.

$$\boldsymbol{v}_t = \eta \boldsymbol{E}_{\mathcal{R}}[x_t] + (1-\eta)\boldsymbol{u}_t \cdot \boldsymbol{G} \tag{4}$$

3.5 Enhancements

We enhance the model without harming its approximation to the FST with the following techniques.

Non-linearity We apply the $\tanh$ activation function to $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$. For example, the third formular of Eqa. 1 becomes: $\boldsymbol{\alpha}_t = \tanh\left((\boldsymbol{g} \cdot \boldsymbol{D}_t^T) \circ \boldsymbol{o}^T\right)$. Note that $y = \tanh(x)$ is close to $y = x$ when $x \in [-1, 1]$, so we find that our model still approximates the FST very well. As will be shown later, when the model is trained with labeled data, applying $\tanh$ leads to better performance.

Dummy States The number of FST states K is decided by the REs. We may introduce K' additional states to improve the capacity of the model. We achieve that by padding the parameter matrices. For example, we pad $\boldsymbol{D}_S \in \mathbb{R}^{K \times R}$ with a random matrix $\boldsymbol{D}'_S \in \mathbb{R}^{K' \times R}$, where $\boldsymbol{D}'_S \sim \mathcal{N}(0, 1e^{-5})$. The padding numbers are so small that these new states can be seen as being isolated with no transition and hence do not interfere with the FST inference. However, transitions from and to these states will be automatically established after training, which often improves the model performance.

Gated Variants We follow (Jiang et al., 2020) and add an update gate $\boldsymbol{z}_t$ and a reset get $\boldsymbol{r}_t$ into the forward and backward score computation. We call it *FSTGRU*. We close the gates initially by setting a big bias term so that our model still approximates the FST. After training, the gates would be utilized and lead to better model performance. We show details in Appendix D.

Algorithm 3:

Input: $\boldsymbol{c}_t, \tau, i$ (index of label $l_\diamond$).
$\boldsymbol{c}'_t = \text{priority}(\boldsymbol{c}_t)$
$\boldsymbol{c}'_t[i] = \min(\boldsymbol{c}'_t[i], \tau)$
$k = \arg\max(\boldsymbol{c}_t)$
$l_t \leftarrow L[k]$
if l_t *is* $l_\diamond$ **then**
  $l_t \leftarrow O$
Output: l_t.

Algorithm 4:

Input: $\boldsymbol{c}_1, \cdots, \boldsymbol{c}_m, \tau, i$.
for $t \leftarrow 1$ **to** m **do**
  $\boldsymbol{c}'_t = \text{priority}(\boldsymbol{c}_t)$
  $\boldsymbol{c}'_t[i] = \min(\boldsymbol{c}_t[i], \tau)$
$l_1, \cdots, l_m \leftarrow \text{viterbi}(\boldsymbol{c}'_1, \cdots, \boldsymbol{c}'_m)$
if l_t *is* $l_\diamond$ **then**
  $l_t \leftarrow O$
Output: $l_1, \cdots, l_m$.

3.6 Decoding

FST inference produces a label score vector $\boldsymbol{c}_t$ at each position t. Here we show how to output labels from the score vectors.

Priority We first feed the label scores at each position into a priority layer to resolve conflicts when different groups capture the same word. The priority relations between slot labels are specified by human experts and can be represented as a set of logic rules. We turn the rules into soft logic computation that can be implemented with an MLP which takes $\boldsymbol{c}_t$ as input and outputs an updated vector $\boldsymbol{c}'_t \in \mathbb{R}^L$. We show details in Appendix E.

FSTRNN-Softmax Before decoding from $\boldsymbol{c}'_t$, we introduce a fixed threshold $\tau \in (0, 1)$ (set to 0.1 by default) to handle the wildcard label $l_\diamond$. Intuitively,

$l_\diamond$ represents unsureness, so we only choose it when the scores of all the other labels are below τ. In that case, we deem the word is not captured by any groups and hence output label O. In all the other cases, we output the highest-scoring label. We show the decoding step in Algorithm 3.

FSTRNN-CRF Linear chain CRF is widely used in sequence labeling decoding. It can be easily integrated into our model. We first handle $l_\diamond$ similarly using threshold τ. We then regard label scores as CRF emission scores and use the Viterbi algorithm to produce the output sequence. Finally, we again change $l_\diamond$ to O. We initialize the CRF transition probabilities $p(y_t|y_{t-1})$ to $\frac{1}{L}$ so that initially we obtain the same output sequence as *FSTRNN-Softmax*. We show the decoding step for *FSTRNN-CRF* in Algorithm 4.

3.7 Training Using Label Data

To approximate REs, we initialize the model parameters: $\boldsymbol{E}_\mathcal{R}, \boldsymbol{D}_S, \boldsymbol{D}_E, \boldsymbol{O}, \boldsymbol{\alpha}_0, \boldsymbol{\beta}_m$ from the corresponding i-FST. But unlike REs, our model can be trained using labeled data to further improve its performance. For *FSTRNN-Softmax*, we optimize the cross-entropy loss at each position. For *FSTRNN-CRF*, we optimize the classic CRF loss. We optimize the loss functions using Adam (Kingma and Ba, 2014). As $l_\diamond$ is a dummy label that does not appear in the training data, its score will be automatically reduced over training, which implies we learn to remove unsureness of the original RE.

4 Experiments

4.1 Experimental Settings

Datasets and REs We perform our experiments on three SF datasets involving 2 languages and 2 domains: ATIS (Hemphill et al., 1990), ATIS-ZH (Mansour and Haider, 2021), and SNIPS (Coucke et al., 2018). ATIS contains English queries for airline service and ATIS-ZH is the Chinese version of ATIS. SNIPS is a collection of English queries to voice assistants. It has a large vocabulary and is more complex than ATIS. Both ATIS and SNIPS are widely adopted for evaluating SF models. For each dataset, we ask an RE expert to write RE rules. We show the statistics of the datasets and REs in Table 2. We leave more details of RE writing in Appendix F.

Baselines We choose *Bi(RNN/GRU/LSTM)-(Softmax/CRF)* (Huang et al., 2015) as base models for experiments with static word embeddings, and use *BERT-base-uncased*[1] + *(Softmax/CRF)* (Chen et al., 2019a) as base models for experiments with contextualized word embeddings. When using BERT, we represent each word using the last BERT hidden state of the first sub-token and feed these hidden states into a linear layer. These baseline methods are widely used for SF. We also compare previous methods of enhancing these base models with REs. Luo et al. (2018) use REs as additional input features (+*i*), use REs to adjust output logits (+*o*) or apply both of them (+*io*)[2]. We also compare two knowledge-distillation-based algorithms that treat RE results as a teacher to help the learning of the base model. One is the classic knowledge distillation method proposed by Hinton et al. (2015) (+*kd*), and the other combines knowledge distillation with posterior regularization (Hu et al., 2016) (+*pr*).

Training Settings When using static word embeddings, we use 100d GloVe (Pennington et al., 2014) embedding for ATIS and SNIPS and use 300d FastText (Bojanowski et al., 2017) embedding for ATIS-ZH; we also fix word embeddings $\boldsymbol{E}_w, \boldsymbol{E}_\mathcal{R}$ during training. When using contextual word embeddings, we finetune BERT for both our methods and the baselines. Our methods have comparable numbers of trainable parameters to those of the baselines. We show the parameter numbers and hyper-parameter tuning of our methods and baselines in Appendix G and H. We compare our methods and the baselines on zero-shot, 10-shot, 50-shot, 10%, and 100% of the training data. We report averaged results from four runs with different random seeds for data sampling and parameter initialization.

4.2 Evaluation

We report the micro F1 scores of our methods and baselines on various training settings. We leave the *BiLSTM* results in Appendix J because *BiGRU* is slightly better. We do the same for +*io* (worse than +*i*).

4.2.1 With Static Word Embeddings

Zero-shot We show the zero-shot performance for models with non-contextual word embeddings in Table 3. Results show that our methods reach

[1] https://huggingface.co/bert-base-uncased

[2] As our model does not have an attention layer, we do not compare our model with their attention based method.

	#Train	#Vocab	#Slots	#RE	Time (h)	K	K_1	Precision	Recall	F1
ATIS	3982	943	83	50	6	114	126	90.34	60.35	72.36
ATIS-ZH	3982	1072	83	34	4	65	94	92.52	63.36	75.21
SNIPS	13084	12134	39	25	8	94	104	85.33	37.37	51.98

Table 2: Statistics of the datasets and REs. For each dataset, we show the number of training samples, vocabulary size and slot number. For the REs of each dataset, K is the state number of the FST converted from the REs and K' is the state number of the corresponding i-FST. We also show the time spent in writing the REs and the SF performance of the REs.

	Model	**ATIS**	**ATIS-ZH**	**SNIPS**
Softmax	*FSTRNN*	**73.10**	74.87	52.02
	FSTGRU	**73.10**	74.87	52.02
	BiGRU+(kd/pr/none)	1.76	1.91	0.69
	BiGRU+i	0.17	0.29	0.48
	BiGRU+o	11.70	22.88	10.49
CRF	*FSTRNN*	**73.10**	74.87	**52.02**
	FSTGRU	**73.10**	74.87	**52.02**
/	*RE*	72.36	**75.21**	51.98

Table 3: Zero-shot results (F1) with static word embeddings.

Model	SNIPS 0	10	50	10%	100%
BERT+FSTRNN	**52.28**	**52.39**	**52.52**	89.07	95.67
BERT+FSTGRU	**52.28**	**51.84**	**52.75**	88.67	95.79
BERT	-	24.79	47.68	89.60	95.60
BERT+i	-	24.78	50.57	90.34	95.71
BERT+o	-	24.92	46.61	89.91	95.75
BERT+kd	-	27.06	48.30	89.81	**96.00**
BERT+pr	-	24.79	47.38	**90.50**	95.57
BERT+FSTRNN+CRF	**52.28**	**52.48**	**52.51**	90.05	95.45
BERT+FSTGRU+CRF	**52.28**	**51.57**	**52.28**	89.43	**96.23**
BERT+CRF	-	25.17	49.96	90.54	95.99
BERT+CRF+i	-	23.47	49.47	89.97	95.71
BERT+CRF+o	-	25.56	50.06	90.52	95.88
BERT+CRF+kd	-	25.42	48.95	90.07	95.66
BERT+CRF+pr	-	25.17	49.96	**90.61**	96.10

Table 4: Zero-shot, low-resource and rich-resource results (F1) for BERT-enhanced models on SNIPS. A redder color indicates better F1. '-' indicates almost random performance.

comparable or even better results in comparison with the original REs without any training and perform much better than all the other baselines. Because *+kd, +pr* are knowledge-distillation-based methods, they are the same as *+none* without any training. The base models and their *+i, +kd, +pr* enhancements perform at the random guessing level, while *+o* is better but still significantly inferior to the original REs. *FSTRNN/FSTGRU* with *Softmax/CRF* perform almost the same because of our initialization strategy described in Sec. 3.5 and Sec. 3.6. sometimes outperform the original REs because of randomness and external word embedding incorporation when $\eta < 1$. We give an analysis of the impact of η in Appendix I.

Few-shot Can our model maintain the lead over the baselines and be trained to outperform the original REs on low-resource settings? The 10-shot and 50-shot results in Table 5 give a **YES** answer. On the few-shot settings, our methods outperform all baselines (including REs). This is because, on one hand, our models have a better starting point derived from the REs in comparison with the neural baselines; and on the other hand, our models can learn to improve using labeled data in comparison with the RE baseline. With 50 shots of the training data, our methods outperform the REs by +2.3 F1 on ATIS and around +1.0 F1 on ATIS-ZH and SNIPS. The enhancement methods, especially *+i*, are effective in low-resource settings and can improve the base models with RE information. *CRF* and gates do not help our methods and baselines because of the lack of training data.

Rich-resource We show the results with 10% and 100% of the training data in Table 5. With static word embeddings, our methods still perform the best with 10% of the training data on ATIS and ATIS-ZH and remain competitive with 100% of the training data on all three datasets. The baselines perform reasonably well with sufficient training samples. However, there does not exist a single baseline that consistently performs the best.

Our methods underperform the baselines on SNIPS 10%, which might be caused by the complexity of the SNIPS dataset and the quality of the RE rules (which have 85 precision compared to 90+ for ATIS and ATIS-ZH). We also observe that *CRF* and gates significantly improve the performance of our methods and baseline without BERT.

	Model	ATIS				ATIS-ZH				SNIPS				Average over Datasets			
		10	50	10%	100%	10	50	10%	100%	10	50	10%	100%	10	50	10%	100%
Softmax	***FSTRNN***	**74.59**	**74.94**	**85.43**	93.82	**75.09**	**75.25**	**82.25**	89.32	**51.94**	**52.84**	78.14	**90.15**	**67.21**	67.68	**81.94**	**91.10**
	BiRNN	57.11	65.80	80.93	**94.30**	63.56	65.07	81.90	**89.89**	17.03	39.37	77.58	87.62	45.90	56.75	80.14	90.60
	BiRNN+i	59.24	69.29	82.25	93.80	65.72	70.84	81.64	89.64	21.60	43.68	**79.45**	88.47	48.86	61.27	81.11	90.64
	BiRNN+o	54.64	66.44	80.63	93.91	64.89	65.79	81.29	89.24	16.87	39.75	76.75	87.95	45.47	57.33	79.56	90.37
	BiRNN+kd	54.21	65.45	81.44	94.18	63.31	65.10	81.80	89.77	17.60	39.56	78.47	88.83	45.04	56.70	80.57	90.93
	BiRNN+pr	55.21	68.28	81.13	93.91	63.56	65.07	81.90	89.73	17.85	39.56	77.50	88.13	45.54	57.64	80.18	90.59
	FSTGRU	**74.59**	**74.94**	**86.89**	94.74	**75.85**	**76.19**	**82.80**	90.50	**52.05**	**52.75**	80.50	90.92	**67.50**	**67.96**	**83.40**	92.05
	BiGRU	52.80	67.69	81.25	94.98	63.62	67.16	81.25	90.28	16.22	41.17	80.51	90.85	44.22	58.67	81.00	92.03
	BiGRU+i	57.68	69.87	83.11	94.63	*64.55*	71.96	82.02	90.16	20.33	44.12	**81.17**	90.70	47.52	61.99	82.10	91.83
	BiGRU+o	52.67	66.97	80.73	94.93	63.29	68.54	80.90	90.13	16.84	40.56	80.44	91.05	44.27	58.69	80.69	92.03
	BiGRU+kd	53.49	67.23	80.99	**95.04**	63.90	67.23	81.36	**90.70**	17.85	41.44	80.16	**91.40**	45.08	58.63	80.84	**92.38**
	BiGRU+pr	52.77	67.69	81.25	94.98	63.35	67.16	81.25	90.28	17.49	41.30	79.94	91.19	44.53	58.72	80.81	92.15
CRF	***FSTRNN***	**74.61**	**74.76**	**85.94**	94.09	**76.08**	**75.92**	**82.92**	90.07	**51.77**	**52.83**	80.77	91.78	**67.49**	**67.83**	**83.21**	**91.98**
	BiRNN	55.04	70.75	82.06	94.30	62.96	67.04	82.82	89.93	17.19	40.47	80.21	90.21	45.07	59.42	81.70	91.48
	BiRNN+i	58.25	69.37	83.84	94.02	65.75	71.40	82.68	89.59	22.18	45.26	**81.90**	**92.24**	48.73	62.01	82.81	91.95
	BiRNN+o	55.26	67.88	83.77	**94.32**	61.47	67.21	82.42	**90.13**	16.81	40.63	79.96	90.45	44.51	58.58	82.05	91.64
	BiRNN+kd	54.93	69.08	82.46	93.58	62.92	67.04	82.84	89.71	17.34	40.48	80.31	90.33	45.06	58.87	81.87	91.21
	BiRNN+pr	56.16	68.02	82.77	93.58	62.96	67.04	82.84	89.64	17.30	40.23	80.47	90.74	45.47	58.43	82.03	91.32
	FSTGRU	**74.61**	**74.76**	**86.50**	95.00	**75.85**	**75.92**	**83.48**	90.73	**52.05**	**53.01**	81.98	**93.17**	**67.50**	**67.89**	**83.99**	**92.97**
	BiGRU	54.43	67.22	79.11	94.66	64.27	68.72	82.71	90.55	18.13	42.47	82.88	92.77	45.61	59.47	81.57	92.66
	BiGRU+i	57.57	70.67	84.44	94.72	64.20	71.43	83.39	90.45	20.76	46.34	**83.30**	92.94	47.51	62.81	83.71	92.70
	BiGRU+o	54.40	67.39	83.24	95.02	63.12	69.27	82.49	90.48	17.40	41.64	82.82	92.49	44.97	59.43	82.85	92.66
	BiGRU+kd	53.31	68.14	82.17	95.22	62.12	68.72	82.52	90.52	17.06	42.47	83.38	92.70	44.17	59.78	82.69	92.81
	BiGRU+pr	53.41	68.34	82.15	**95.39**	62.46	68.72	82.71	**90.75**	17.01	42.47	83.21	92.75	44.29	59.84	82.69	92.96

Table 5: Low- and rich-resource results (F1) with static word embeddings. Redder colors indicate better F1s.

4.2.2 With BERT

We show the zero-shot, low-resource, and rich-resource performance of BERT-enhanced methods on the SNIPS dataset in Table 4. We choose between the two initialization strategies of $\boldsymbol{E}_w$, **Aggregate** and **Random**, based on the development set performance.

Zero-shot The experimental results show that our methods can still approximate the REs well, reaching 52.25 F1 (+0.27 compared to original REs) without any training data.

Low-resource In the low-resource setting (10 and 50 training samples), our methods have minor or even no improvement over the zero-shot setting. On the other hand, the BERT-enhanced baselines perform much better compared to the non-BERT baselines, but they are still far behind our methods.

Rich-resource The experimental results on 10% and 100% of training data show that both our methods and the baselines have a large performance gain compared with the non-BERT setting (around +8% and +4%). Our methods are still competitive with the baselines, especially when using 100% of training data, which shows the ability of our methods to utilize pretrained contextual word embeddings such as BERT.

In addition to the SNIPS dataset, we also test our methods with BERT on the ATIS dataset. While our method again beats the baselines on the zero-shot and 10-shot settings and is competitive on the 100% setting, it falls behind the baselines on the 50-shot and 10% settings, especially when using *CRF*. We speculate that the ATIS REs may not be very compatible with BERT-enhanced models and hence our model may require more training data to move away from the RE-based initialization. We leave more detailed analyses for future investigation.

5 Analysis

Ablation Study We conduct an ablation study on the 100% training samples on the three datasets in Table 6. The ablation results show that our various enhancements indeed improve the model performance, especially the `tanh` nonlinearity and external word embeddings. The randomly initialized *FSTGRU+CRF* performs surprisingly well, which indicates that our model has a strong capacity to learn from data and does not rely much on rules with enough training data.

Utilizing Unlabeled Data We assume we do not have unlabeled data in previous experiments. However, with clean unlabeled data, we can use REs to produce pseudo-labels and use them to train baselines. We compare the results of training *BiGRU+(none/i/o)+CRF* using different amounts of unlabeled data and *FSTRNN* without any training on SNIPS (Fig 4). Results show that we need

100% Training data	ATIS	ATIS-ZH	SNIPS
FSTGRU+CRF	**95.00**	**90.73**	**93.17**
w/o *CRF*	94.63	90.50	90.92
w/o gates	94.09	90.07	91.78
w/o nonlinear	93.34	86.90	90.44
w/ ReLU nonlinear	94.29	89.17	89.16
w/o dummy states	94.61	90.22	92.09
w/o word embed. ($\eta = 1$)	94.56	90.00	90.42
randomly initialized	94.71	90.60	91.73

Table 6: Ablation Study

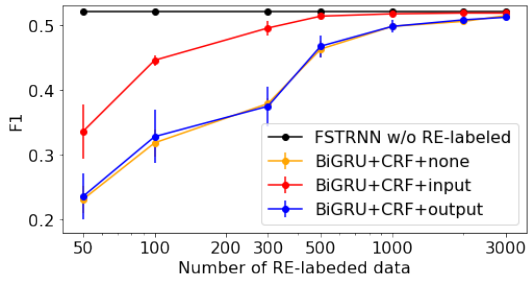


Figure 4: Training baselines with RE-labeled data on SNIPS.

around 1000 unlabeled data for *+i* and 3000 unlabeled data for *+o, +none* to catch up the performance of *FSTRNN* without training.

6 Related Work

Slot Filling Symbolic systems for slot filling are mainly based on REs (Liu and Zhao, 2012; Larson et al., 2020). Neural approaches usually treat slot filling as a sequence labeling problem and solve it using recurrent neural networks and conditional random fields (Mesnil et al., 2014; Yao et al., 2014; Liu and Lane, 2016). More recently, many studies focus on the joint learning of slot filling and intent detection (Goo et al., 2018; Chen et al., 2019b).

Combining Symbolic Rules with Neural Networks The majority of previous work uses the output of rules as additional supervision or input features to neural networks. Luo et al. (2018) use RE outputs as additional input features or use them to guide the attention or softmax logits of neural models. Hu et al. (2016) combine knowledge distillation (Hinton et al., 2015) with posterior regularization to distill the results of first-order logic deduction into neural model. Li and Srikumar (2019) use first-order logic to constrain the output of neural layers. Other methods use symbolic knowledge in pretraining (Rosset et al., 2020; Tian et al., 2020) or use symbolic rules to constrain text generation (Lin et al., 2020; Li and Rush, 2020). Our method is different from these methods in that we directly neuralize the RE symbolic system, instead of having separate symbolic and neural systems. There are also previous works on combining FSTs with neural networks. Rastogi et al. (2016) augment an FST with neural context features extracted by BiLSTMs and apply it to morphological reinflection. Lin et al. (2019) use RNNs to score the accepting paths of FSTs and apply their model to the string transduction tasks. They train the model using importance sampling and decode it using various approximation methods. The main difference of our model from the above methods is that, instead of combining an FST with RNNs, we directly convert an FST into a trainable neural network.

Our work is inspired by Jiang et al. (2020) who neuralize REs for text classification. Apart from the difference in the output forms, our method differs from theirs mainly in that: (1) we target the slot filling task and hence neuralize REs with capturing groups; (2) we convert REs into an FST that produces output sequences, while they convert REs into a finite state automaton (FSA) that decides acceptance of strings; (3) the tensor parameter of an FST has higher order than an FSA, which leads to a more complicated procedure to reduce the computational complexity; (4) our decoding algorithm has to invoke both forward and backward processes, while theirs requires only a forward process.

7 Conclusion and Future Work

In this work, we neuralize a symbolic RE system for slot filling into a trainable neural network model. The model approximates REs well initially and can be trained to improve itself with labeled data. Experiments in various settings show the advantages of our method. To the best of our knowledge, we are the first to neuralize REs into neural networks for the slot filling task.

For future work, we want to explore the possibility of converting an *FSTRNN* back into REs for better interpretability, and investigate more on utilizing BERT.

We believe our methodology can be extended to other tasks such as NER and QA. We also hope our methods can provide insights into theoretical analysis on the relations between neural models and regular languages.

8 Acknowledgement

This work was supported by the National Natural Science Foundation of China (61976139).

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Rishi Bommasani, Kelly Davis, and Claire Cardie. 2020. Interpreting Pretrained Contextualized Representations via Reductions to Static Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4758–4781, Online. Association for Computational Linguistics.

Francisco Casacuberta and Colin De La Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *International Colloquium on Grammatical Inference*, pages 15–24. Springer.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019a. BERT for joint intent classification and slot filling. *CoRR*, abs/1902.10909.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019b. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *CoRR*, abs/1805.10190.

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. JMLR Workshop and Conference Proceedings.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757.

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*.

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.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420, Berlin, Germany. Association for Computational Linguistics.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Chengyue Jiang, Yinggong Zhao, Shanbo Chu, Libin Shen, and Kewei Tu. 2020. Cold-start and interpretability: Turning regular expressions into trainable recurrent neural networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3193–3207, Online. Association for Computational Linguistics.

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

Stefan Larson, Eric Guldan, and Kevin Leach. 2020. Data query language and corpus tools for slot-filling and intent classification data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 7060–7068, Marseille, France. European Language Resources Association.

Tao Li and Vivek Srikumar. 2019. Augmenting neural networks with first-order logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 292–302, Florence, Italy. Association for Computational Linguistics.

Xiang Lisa Li and Alexander Rush. 2020. Posterior control of blackbox generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2731–2743, Online. 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, Minneapolis, Minnesota. Association for Computational Linguistics.

Xiexiong Lin, Weiyu Jian, Jianshan He, Taifeng Wang, and Wei Chu. 2020. Generating informative conversational response using recurrent knowledge-interaction and knowledge-copy. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 41–52, Online. Association for Computational Linguistics.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*.

Fang Liu and Jun Zhao. 2012. Sweat2012: Pattern based english slot filling system for knowledge base population at tac 2012. In *TAC*.

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. *arXiv preprint arXiv:1805.05588*.

Saab Mansour and Batool Haider. 2021. Atis seven languages ldc2021t04. In *Philadelphia: Linguistic Data Consortium, 2021*.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2014. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539.

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.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 623–633, San Diego, California. Association for Computational Linguistics.

Corby Rosset, Chenyan Xiong, Minh Phan, Xia Song, Paul N. Bennett, and Saurabh Tiwary. 2020. Knowledge-aware language model pretraining. *CoRR*, abs/2007.00655.

Yuto Sakuma, Yasuhiko Minamide, and Andrei Voronkov. 2012. Translating regular expression matching into transducers. *Journal of Applied Logic*, 10(1):32–51.

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

Hao Tian, Can Gao, Xinyan Xiao, Hao Liu, Bolei He, Hua Wu, Haifeng Wang, and Feng Wu. 2020. SKEP: Sentiment knowledge enhanced pre-training for sentiment analysis. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4067–4076, Online. Association for Computational Linguistics.

Peifeng Wang, Jialong Han, Chenliang Li, and Rong Pan. 2019. Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7152–7159.

Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. 2014. Spoken language understanding using long short-term memory neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 189–194. IEEE.

Shanshan Zhang, Lihong He, Slobodan Vucetic, and Eduard Dragut. 2018. Regular expression guided entity mention mining from noisy web data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1991–2000, Brussels, Belgium. Association for Computational Linguistics.

A RE with capturing groups to FST outputs BIO tags

This section shows the procedure of turning REs with capturing groups into FST outputs correct BIO tags step by step. By Theorem A.1, we first ignore the capturing group and transformed the REs into an NFA, and then we add output labels for transitions based on capturing groups to make it an FST. Since the method is very pivotal for practical applications, we provide handy tool-kits and examples for writing REs and converting REs into FST in https://github.com/jeffchy/RE2NN-SEQ.

Theorem A.1. *Any FST with input vocabulary Σ and output vocabulary Γ can be viewed as a NFA with input vocabulary $\Sigma \times \Gamma$ where $\times$ denotes the cartesian product.*

Theorem A.2. *If r is a regular expression without capturing group, then it can be converted into an nondeterministic finite automata with epsilon transitions (ϵ-NFA) .*

Theorem A.3. *Any ϵ-NFA can be turned into an NFA.*

Theorem A.4. *Any NFA can be turned into a deterministic finite automaton (DFA).*

Step 1: Given a RE with k capturing groups with group names $l_1, \cdots, l_k$, we first inject the empty string ϵ to split the sub-expressions that with and without capturing groups. For example, the RE: $w_\diamond^*$ *from* $\left[w_\diamond^*\right]\langle$ *fr.city* $\rangle$ *to* $w_\diamond^*$ becomes $\left(w_\diamond^*\right.$ *from*$\left.\right)$ ϵ $\left(\left[w_\diamond^*\right]\langle\right.$ *fr.city* $\left.\rangle\right)$ ϵ $\left(\right.$*to* $\left.w_\diamond^*\right)$.

Step 2: We ignore the capturing groups and perform the Thompson's construction on the whole RE and get the ϵ-NFA (Theorem A.2). However, during the construction, we mark the states and edges of each sub-NFA corresponds to each capturing group and denote them as $\mathcal{S}_1, \cdots, \mathcal{S}_k$, and $E_1, \cdots, E_k$, where $E_k = \{(s_i, s_j) | s_i, s_j \in \mathcal{S}_k\}$. We also use $\mathcal{S}_0$ and E_0 to denote the states and

edges outside the capturing groups. As we separate sub-expressions with ϵ, due to the concatenation rule of Thompson's construction, we have $\mathcal{S}_i \cap \mathcal{S}_j = \varnothing$, $E_i \cap E_j = \varnothing$ for $i \neq j$ and $0 \leq i, j \leq k$.

Step 3: Assign $l_\diamond$ as the output of edges in E_0 whose input is not ϵ.

Step 4: Make sure for each sub-NFA, the start state has no outgoing ϵ-edges and incoming edges. Then assign BIO-tags to the new sub-NFA. We show the algorithm details in Algorithm 5.

Algorithm 5: Step 4

```
1  input: sub-NFAs, ⟨S_1, E_1, ⟩ ··· , ⟨S_k, E_k, ⟩
2  for i ← 1 to k do
3      Perform the ϵ-elimination to the i-th sub-NFA to
         get the new states and edges S'_i, E'_i.
4      Find the start state of the new sub-NFA. s_0 ∈ S'_i.
5      Create a new state s'.
6      for s_j ∈ S'_i do
7          if edge (s_j, s_0) ∈ E'_i then
8              Remove edge (s_j, s_0);
9              Create edge (s_j, s');
10         end
11         if edge (s_0, s_j) ∈ E'_i then
12             Create edge (s', s_j);
13         end
14     end
15     Upon getting the updated sub-NFA, assign B-l_k
         to the outgoing edges from the start state and
         I-l_k to all other edges.
16 end
```

Step 5: We do the ϵ-elimination again to eliminate the ϵs in E_0 and the ϵs we add to separate the capturing groups at the beginning. Then we convert it into a DFA (Theorom A.3) and minimize it using the Hopcroft algorithm (Hopcroft, 1971).

As most steps are either trivial or based on a provided theorem, we only prove the correctness of the following lemma.

Lemma A.5. *Given an NFA, Algorithm 5 produces an equivalent NFA whose start state has no outgoing ϵ-edges and incoming edges.*

Proof. Given an NFA, the algorithm first performs ϵ-elimination (line 4) to the NFA, by Theorem A.4, the new NFA has no ϵ-edges and are equivalent to the original one. Later steps (lines 5 to 14) will not produce ϵ edges. Therefore the start state has no outgoing ϵ-edges.

Then we prove that the algorithm from line 4 to line 13 produces an equivalent NFA whose start state has no incoming edges. We denote the original NFA $\mathcal{A}$ and the converted NFA $\mathcal{A}'$, and $L(\mathcal{A})$ denotes the language set of $\mathcal{A}$. We prove that $\mathcal{A}$ is equivalent to $\mathcal{A}'$ by showing $L(\mathcal{A}) \subset L(\mathcal{A}')$ and $L(\mathcal{A}') \subset L(\mathcal{A})$. (1) $L(\mathcal{A}) \subset L(\mathcal{A}')$. The paths that via the original incoming edges of the start state will go through the new state because we change the destinations of these incoming edges from the start state to the new state (lines 8 and 9). They will reach the final state because we copy all outgoing edges of the start state to the new state. (line 12). For the paths that do not go by these incoming edges, they will not go through the new state and reach the final states. (2) $L(\mathcal{A}') \subset L(\mathcal{A})$. Similarly, for the paths go through the new states will go through the incoming edges of the start state in the original NFA, and as the outgoing edges of the start state and the new state are the same, it must reach the final states in the $\mathcal{A}$. For the paths that do not go through the new state, we can remove the new state and corresponding edges. It is the same as the $\mathcal{A}$ removing incoming edges of the start state. Proved.

□

B FST to s-FST

We present the conversion algorithm below (Algorithm 6).

C CP decomposition (CPD)

CPD is also known as tensor rank decomposition. For a Nth-order tensor $\boldsymbol{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_N}$ can be approximated as sum of R rank-1 tensors.

$$
\begin{aligned}
\boldsymbol{T} \approx \hat{\boldsymbol{T}} &= \sum_{i=1}^{R} \boldsymbol{f}_1^{(i)} \otimes \cdots \otimes \boldsymbol{f}_N^{(i)} \\
\hat{\boldsymbol{T}}_{(1)} &= (\boldsymbol{F}_2 \odot \boldsymbol{F}_3 \odot \cdots \odot \boldsymbol{F}_N)\boldsymbol{F}_1^T \\
\boldsymbol{F}_i &= \left[\boldsymbol{f}_1^{(i)} \cdots \boldsymbol{f}_N^{(i)}\right], \boldsymbol{F}_i \in \mathbb{R}^{d_i \times r}
\end{aligned}
\tag{5}
$$

$\otimes$ denotes the outer product, $\odot$ denotes the Khatri-Rao product and $\hat{\boldsymbol{T}}_{(1)}$ denotes the mode-1 unfolding of tensor $\hat{\boldsymbol{T}}$. If the rank R is large enough (e.g. larger than the tensor rank of $\boldsymbol{T}$), the decomposition can be exact.

We also apply the tricks mentioned in Jiang et al. (2020)'s Appendix B to speed up the decomposition and normalize the decomposed matrices.

Algorithm 6: FST to s-FST.

Input: $\mathcal{T} = \langle\, Q, \Sigma, \Gamma, Q_I, Q_F, \Omega \,\rangle$.
Output: a new FST $\mathcal{T}'$
for $q \in Q$ **do**
  Partition transitions to q based on their all possible output $\mathcal{O}$ and form $k = |\mathcal{O}|$ sets $E_1, \cdots E_k$ where $E_i = \{\langle q_j, q, \sigma, o_i\rangle \,|q_j \in Q, \sigma \in \Sigma, o_i \in \mathcal{O}\}$
  Get the outgoing transitions of q, denote as E_O
  if $k > 1$ **then**
    for $j \leftarrow 2$ **to** k **do**
      create a new state q';
      for $\langle q_m, q, \sigma_m, o_j\rangle \in E_j$ **do**
        remove transition $\langle q_m, q, \sigma_m, o_j\rangle$;
        create transition $\langle q_m, q', \sigma_m, o_j\rangle$;
        for $\langle q, q_n, \sigma_n, o_n\rangle \in E_O$ **do**
          if $q \neq q_n$ **then**
            create transition $\langle q', q^n, \sigma_n, o_n\rangle$;
          else if $q = q_n$ ***and*** $o_j = o_n$ **then**
            create transition $\langle q', q', \sigma_n, o_n\rangle$;
          end
        end
      end
    end
  end
end

D Gated Variants

We add an update gate $\boldsymbol{z}_t$, and a reset gate $\boldsymbol{r}_t$ into the forward and backward score computation of FSTRNN and get **FSTGRU**

Take the forward score as an example. We compute the $\boldsymbol{z}_t$ and $\boldsymbol{r}_t$ using $\boldsymbol{v}_t$ and $\boldsymbol{\alpha}_{t-1}$:

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{\sigma}(\boldsymbol{W}_z \boldsymbol{v}_t + \boldsymbol{U}_z \boldsymbol{\alpha}_{t-1} + \boldsymbol{b}_z) \\ \boldsymbol{f}_r &= \boldsymbol{\sigma}(\boldsymbol{W}_r \boldsymbol{v}_t + \boldsymbol{U}_r \boldsymbol{\alpha}_{t-1} + \boldsymbol{b}_r) \end{aligned} \tag{6}$$

where $\boldsymbol{\sigma}$ is the sigmoid activation, and $\boldsymbol{W}_z, \boldsymbol{W}_r, \boldsymbol{U}_z, \boldsymbol{U}_r, \boldsymbol{b}_z, \boldsymbol{b}_r$ are trainable parameters of gates. We apply these gates to the forward score computation and Eqa 1 becomes:

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{t-1} &= (1 - \boldsymbol{r}_t) \circ \boldsymbol{\alpha}_0 + \boldsymbol{r}_t \circ \boldsymbol{\alpha}_{\boldsymbol{t-1}} \\ \boldsymbol{g} &= (\hat{\boldsymbol{\alpha}}_{t-1} \cdot \boldsymbol{D}_f) \circ \boldsymbol{v}_t \\ \boldsymbol{\alpha}_t &= (\boldsymbol{g} \cdot \boldsymbol{D}_t^T) \circ \boldsymbol{o}^T \\ \boldsymbol{\alpha}_t &= (1 - \boldsymbol{z}_t) \circ \boldsymbol{\alpha}_{t-1} + \boldsymbol{z}_t \circ \boldsymbol{\alpha}_t \end{aligned} \tag{7}$$

We initialize $\boldsymbol{W}_z, \boldsymbol{W}_r, \boldsymbol{U}_z, \boldsymbol{U}_r$ randomly using Xavier normal (Glorot and Bengio, 2010), and bias terms $\boldsymbol{b}_z, \boldsymbol{b}_r$ as a big integer (e.g. 10). Therefore, we close the gates at the beginning to make our model still approximate the FST, and after training, the performance can be improved because of the gating mechanism.

E Priority

We show how we resolve priority problems here with an example. For the sentence *I love action movie*, the following RE can capture *movie* and *action movie* at the same time.

$$w_\diamond^\star \ \textit{(movie|action movie)}\langle \textit{movie} \rangle \ w_\diamond^\star$$

So the word *action* can be tagged as *B-movie* or *O*, the word *movie* can be tagged as *B-movie* or *I-movie*. Assume that we prefer longer captured text, we can define the following priority: *B-movie* > *O*, *I-movie* > *O*, *I-movie* > *B-movie*, here the $>$ operator means that the label on the left has higher priority compared to label on the right and these priority can be written into logic. For example, the priority *B-movie* > *O* can be expressed as: $\textsc{Match}(O) \wedge \neg\textsc{Match}(\textit{B-movie}) \Rightarrow \textsc{Label}(O)$. We can encode this logic into our model using soft logic. Let L_a, L_a be proposition symbols whose soft truth score are a, b, the soft version of $\neg L_a$ is $1 - a$, and the soft version of $L_a \wedge L_b$ is $\max(0, a + b - 1)$. So the soft version of is $\max(0, b - a)$, which can be easily implemented using a $L \times L$ matrix and a ReLU nonlinear. The priority layer is optional; as in our experiments, conflicts seldom occur.

F Writing REs

We try to mimic the RE annotating procedure in real applications. In industry, REs for slot filling applications are written by RE experts with domain knowledge (or with the help of a domain expert).

In our experiments, we ask an RE expert to write RE rules for these datasets. As the expert does not

have domain knowledge and is not familiar with the data, we follow the method of (Luo et al., 2018) to 'teach' the expert domain knowledge. More specifically, we sample 40-shot of training data and ask the expert to write REs to capture them. It takes the expert around 6 hours to write rules for ATIS, 8 hours for SNIPS. And 4 hours for ATIS-ZH. Experts usually have domain knowledge in reality. Hence the writing process can be further accelerated, and less or even no examples are required. We also show examples of Written REs for each dataset in Table 7.

G Number of trainable parameters

We set the word embedding dim $D = 100$, the rank $R = 150$, the number of FST states $K = 150$, the number of slot labels $L = 50$, and the number of hidden states $H = 100$ to calculate the number of trainable parameters of our method and baselines. We show the results in Table 8.

H Hyper-parameter tuning

We tune the hyper-parameter of our methods and baselines on the development set using grid search. We report the grids in the following Table 9.

I Analysis on η

We show how the η influences the performance of our model on SNIPS in zero-shot and rich resource settings (Fig 5). The best η on zero-shot settings is 0.9 instead of 1, which means integrating some word information can improve the rule without any training. After training with 100% of training data, a small η (e.g. 0.1) performs best because word embedding can help model learning with sufficient data.

J Full results

We show the full results with *BiLSTM* and *+io* baselines, and standard deviations of three datasets in Table 10, Table 11 and Table 12.

Figure 5: η v.s. performance on the zero-shot and 100% train settings in SNIPS dataset.

	RE examples
ATIS	$w_\diamond^*$ from $[w_\diamond^*]\langle$ *fr.city* $\rangle$ to $w_\diamond^*$ $w_\diamond^*$ (arrive|arrives) $w_\diamond^*$ $[$(monday|tuesday|$\cdots$|sunday)$]\langle$ *arrive.day_name* $\rangle$ $w_\diamond^*$
ATIS-ZH	$w_\diamond^*$ 从 $[w_\diamond^*]\langle$ *fr.city* $\rangle$ (到|抵达|飞往) $w_\diamond^*$ $w_\diamond^*$ $[$(周一|周二|$\cdots$|周日)$]\langle$ *arrive.day_name* $\rangle$ $w_\diamond^*$ (到|抵达|飞往) $w_\diamond^*$
SNIPS	$w_\diamond^*$ $[$(stars|points)$]\langle$ *rating.units* $\rangle$ $w_\diamond^*$ $w_\diamond^*$ (add|put) $w_\diamond^*$ to $[$my$]\langle$ *playlist.owner* $\rangle$ $[w_\diamond\{1,3\}]\langle$ *playlist* $\rangle$ playlist $w_\diamond^*$

Table 7: RE examples for three datasets. REs given for ATIS and ATIS-ZH are corresponded one to one. $w_\diamond\{1,3\}$ means any word can appear 1 to 3 times.

Model (*Softmax*)	Formular	# Parameter
FSTRNN	$DR + 2KR + KL$	67500
FSTGRU	$DR + 4KR + 2RR + KL$	157500
BiRNN	$2DH + 2HH + HL$	45000
BiGRU	$6DH + 6HH + HL$	125000
BiLSTM	$8DH + 8HH + HL$	165000

Table 8: Number of trainable parameters

Model	Hyper-parameter	Grid
FSTRNN	learning rate	[3e-4, 1e-3, 3e-3]
	# dummy states	[0, 30, 60, 90]
	η	[0.1, 0.3, 0.5, 0.7, 0.9]
BiLSTM	learning rate	[1e-3, 2e-3, 5e-3, 1e-2]
	# hidden states	[100, 150, 200]
+kd	temperature	[1, 3, 5]
+kd, +pr	weight for student loss	[0.3, 0.8]
+pr	annealing term	[1.0, 0.98]

Table 9: Hyper-parameter tuning.

Model		0 shot		10 shot		50 shot		10%		100%	
		mean	std	mean	std	mean	std	mean	std	mean	std
Softmax	*FSTRNN*	73.07	0.79	74.59	0.73	74.94	0.38	85.43	1.06	93.82	0.12
	FSTGRU	73.07	0.79	74.59	0.73	74.94	0.38	86.89	1.08	94.63	0.13
	GRU	1.76	3.04	52.80	3.71	67.69	2.47	81.25	0.97	94.98	0.37
	GRU+i	0.17	0.26	57.68	3.41	69.87	2.55	83.11	0.38	94.63	0.19
	GRU+o	11.70	7.30	52.67	4.28	66.97	2.72	80.73	2.26	94.93	0.27
	GRU+io	11.04	5.35	58.12	3.00	69.99	2.90	82.84	0.75	94.59	0.32
	GRU+kd	1.76	3.04	53.49	2.85	67.23	3.05	80.99	1.00	95.04	0.39
	GRU+pr	1.76	3.04	52.77	3.61	67.69	2.47	81.25	0.97	94.98	0.37
	LSTM	0.31	0.36	56.22	0.25	66.34	3.81	80.49	1.26	94.46	0.11
	LSTM+i	0.36	0.44	57.54	4.67	69.48	2.70	81.24	0.69	94.51	0.13
	LSTM+o	12.93	6.53	53.40	3.23	67.12	3.13	80.09	1.70	94.57	0.27
	LSTM+io	11.19	6.27	57.56	3.31	69.99	1.85	81.00	1.53	94.56	0.07
	LSTM+kd	0.31	0.36	54.70	1.89	66.94	2.80	80.67	0.67	94.66	0.30
	LSTM+pr	0.31	0.36	53.84	2.84	66.42	3.49	80.39	0.75	94.46	0.11
	RNN	0.47	0.51	57.11	4.39	65.80	3.34	80.93	1.14	94.30	0.47
	RNN+i	1.10	1.77	59.24	1.99	69.29	2.38	82.25	0.89	93.80	0.44
	RNN+o	13.83	7.15	54.64	1.98	66.44	3.22	80.63	1.32	93.91	0.54
	RNN+io	12.21	7.40	58.94	2.77	69.52	2.36	81.86	1.07	93.74	0.24
	RNN+kd	0.47	0.51	54.21	3.31	65.45	3.17	81.44	0.72	94.18	0.47
	RNN+pr	0.47	0.51	55.21	2.73	68.28	4.71	81.13	0.58	93.91	0.52
CRF	*FSTRNN*	73.10	0.88	74.61	0.49	74.76	0.60	85.94	0.67	94.74	0.39
	FSTGRU	73.10	0.88	74.61	0.49	74.76	0.60	86.50	0.83	95.00	0.40
	GRU	0.63	0.76	54.43	4.76	67.22	4.18	79.11	1.22	94.66	0.02
	GRU+i	0.17	0.17	57.57	6.61	70.67	3.18	84.44	0.25	94.72	0.32
	GRU+o	26.00	15.73	54.40	2.15	67.39	3.16	83.24	1.04	95.02	0.21
	GRU+io	14.89	13.62	58.67	3.83	70.57	3.03	85.25	0.72	95.00	0.17
	GRU+kd	0.63	0.76	53.31	4.44	68.14	2.40	82.17	0.88	95.22	0.29
	GRU+pr	0.63	0.76	53.41	4.39	68.34	2.34	82.15	1.08	95.39	0.33
	LSTM	0.18	0.15	52.19	2.60	65.75	3.89	80.27	0.61	94.91	0.11
	LSTM+i	0.71	0.97	57.69	3.02	70.06	2.52	82.22	1.05	94.44	0.22
	LSTM+o	28.95	14.06	54.90	3.05	66.76	4.39	81.96	0.50	95.08	0.14
	LSTM+io	24.98	19.39	57.68	3.24	69.40	1.95	82.79	1.31	94.47	0.12
	LSTM+kd	0.18	0.15	52.37	2.72	67.28	2.71	81.63	0.71	95.00	0.19
	LSTM+pr	0.18	0.15	52.38	2.72	67.28	2.71	81.44	0.43	95.03	0.27
	RNN	0.05	0.08	55.04	3.04	70.75	1.92	82.06	0.12	94.30	0.30
	RNN+i	0.20	0.21	58.25	3.61	69.37	3.21	83.84	0.66	94.02	0.16
	RNN+o	13.84	11.88	55.26	2.48	67.88	2.35	83.77	0.90	94.32	0.28
	RNN+io	11.57	9.18	57.52	1.42	69.18	3.24	83.87	0.61	94.01	0.14
	RNN+kd	0.05	0.08	54.93	2.95	69.08	1.33	82.46	1.57	93.58	0.00
	RNN+pr	0.05	0.08	56.16	2.95	68.02	2.33	82.77	0.62	93.58	0.29

Table 10: Full ATIS results

Model		0 shot		10 shot		50 shot		10%		100%	
		mean	std	mean	std	mean	std	mean	std	mean	std
Softmax	*FSTRNN*	52.02	0.02	51.94	0.23	52.84	1.18	78.14	0.70	90.15	0.73
	FSTGRU	52.02	0.06	52.05	0.00	52.83	1.36	80.50	0.69	90.92	0.43
	GRU	0.69	0.79	16.22	2.52	41.17	4.11	80.51	0.35	90.85	0.26
	GRU+i	0.48	0.35	20.33	2.87	44.12	3.16	81.17	0.76	90.70	0.57
	GRU+o	10.49	2.73	16.84	2.38	40.56	3.90	80.44	0.59	91.05	0.95
	GRU+io	9.83	3.85	20.60	2.91	45.75	2.90	81.14	0.61	90.83	0.33
	GRU+kd	0.69	0.79	17.85	2.68	41.44	4.29	80.16	0.62	91.40	0.47
	GRU+pr	0.69	0.79	17.49	2.47	41.30	4.41	79.94	0.65	91.19	0.31
	LSTM	0.90	0.84	15.35	3.85	39.96	5.27	78.28	0.71	90.68	0.58
	LSTM+i	1.09	1.17	20.08	3.31	43.22	3.11	79.75	0.79	91.22	0.58
	LSTM+o	13.91	2.00	16.16	1.65	40.28	3.84	77.71	0.50	90.93	0.43
	LSTM+io	13.65	2.69	20.61	3.17	44.53	2.93	80.05	1.61	91.51	0.65
	LSTM+kd	0.90	0.84	18.11	4.33	40.61	4.65	79.63	1.18	91.65	0.49
	LSTM+pr	0.90	0.84	17.15	4.04	39.95	5.09	78.88	0.33	90.99	0.36
	RNN	0.88	0.86	17.03	2.38	39.37	4.59	77.58	0.34	87.62	0.26
	RNN+i	0.48	0.18	21.60	2.36	43.68	3.77	79.45	0.69	88.47	0.78
	RNN+o	7.58	3.35	16.87	2.68	39.75	4.06	76.75	0.73	87.95	0.65
	RNN+io	7.15	2.92	21.51	1.80	44.04	3.39	78.76	0.51	89.29	0.59
	RNN+kd	0.88	0.86	17.60	2.47	39.56	4.10	78.47	0.90	88.83	0.32
	RNN+pr	0.88	0.86	17.85	3.18	39.56	4.49	77.50	1.32	88.13	0.62
CRF	*FSTRNN*	52.02	0.02	51.77	0.57	52.75	1.38	80.77	0.70	91.78	0.26
	FSTGRU	52.02	0.02	52.05	0.00	53.01	1.54	81.98	0.75	93.17	0.67
	GRU	0.28	0.11	18.13	2.89	42.47	4.59	82.88	0.65	92.77	0.36
	GRU+i	0.58	0.39	20.76	2.98	46.34	3.11	83.30	1.31	92.94	0.67
	GRU+o	11.76	4.94	17.40	2.58	41.64	5.16	82.82	0.97	92.49	0.31
	GRU+io	10.13	5.19	19.73	3.60	46.69	3.08	83.33	1.15	92.83	0.48
	GRU+kd	0.28	0.11	17.06	2.73	42.47	4.59	83.38	0.89	92.70	0.53
	GRU+pr	0.28	0.11	17.01	2.74	42.47	4.59	83.21	1.03	92.75	0.72
	LSTM	0.38	0.20	16.23	3.32	40.76	5.78	80.69	0.66	92.17	0.25
	LSTM+i	0.55	0.58	20.75	3.41	46.03	4.72	82.17	0.65	92.68	0.33
	LSTM+o	10.77	2.61	19.76	7.46	40.81	6.34	81.04	1.43	92.54	0.47
	LSTM+io	11.46	6.27	20.67	7.21	44.56	4.39	82.35	0.41	93.09	0.57
	LSTM+kd	0.38	0.20	16.54	3.12	40.76	5.78	82.17	1.00	92.42	0.54
	LSTM+pr	0.38	0.20	16.23	3.32	40.76	5.78	82.18	1.03	92.66	0.47
	RNN	0.31	0.36	17.19	2.76	40.47	4.33	80.21	0.68	90.21	0.54
	RNN+i	0.97	0.90	22.18	2.37	45.26	3.65	81.90	0.34	92.24	0.55
	RNN+o	9.09	4.74	16.81	2.92	40.63	4.68	79.96	1.20	90.45	0.23
	RNN+io	7.74	2.27	21.47	3.28	45.22	3.87	81.35	1.04	90.84	0.73
	RNN+kd	0.31	0.36	17.34	2.80	40.48	4.31	80.31	0.38	90.33	0.05
	RNN+pr	0.31	0.36	17.30	2.59	40.23	4.63	80.47	0.41	90.74	0.60

Table 11: Full SNIPS results

Model		0 shot		10 shot		50 shot		10%		100%	
		mean	std	mean	std	mean	std	mean	std	mean	std
Softmax	*FSTRNN*	74.84	0.06	75.09	0.04	75.25	0.18	82.25	1.05	89.32	0.50
	FSTGRU	74.87	0.06	75.85	0.49	76.19	0.45	82.80	1.13	90.50	0.31
	GRU	1.91	3.48	63.62	3.39	67.16	5.79	81.25	0.95	90.28	0.30
	GRU+i	0.29	0.48	64.55	1.43	71.96	1.31	82.02	0.85	90.16	0.55
	GRU+o	22.88	17.38	63.29	5.74	68.54	4.96	80.90	0.84	90.13	0.45
	GRU+io	20.01	15.93	65.37	2.09	71.54	2.16	82.23	0.74	90.17	0.15
	GRU+kd	1.91	3.48	63.90	3.43	67.23	5.99	81.36	0.87	90.70	0.28
	GRU+pr	1.91	3.48	63.35	3.32	67.16	5.79	81.25	0.95	90.28	0.30
	LSTM	0.34	0.41	64.27	2.68	64.69	2.41	81.69	0.72	90.58	0.82
	LSTM+i	0.74	0.75	65.11	2.62	70.72	1.98	82.87	0.51	89.97	0.32
	LSTM+o	18.66	15.75	64.18	2.38	64.96	1.86	81.68	1.00	89.91	0.49
	LSTM+io	19.79	15.76	65.09	1.60	70.60	3.05	82.82	0.91	89.95	0.46
	LSTM+kd	0.34	0.41	63.73	4.50	65.89	1.77	81.62	0.66	91.15	0.22
	LSTM+pr	0.34	0.41	64.27	2.68	64.69	2.41	81.69	0.72	90.41	0.49
	RNN	0.58	0.84	63.56	2.42	65.07	4.53	81.90	1.10	89.89	0.48
	RNN+i	0.71	1.15	65.72	2.27	70.84	1.04	81.64	1.35	89.64	0.38
	RNN+o	20.61	15.94	64.89	2.73	65.79	3.51	81.29	1.31	89.24	0.27
	RNN+io	21.24	16.63	65.54	2.00	70.92	2.04	81.68	1.29	89.59	0.32
	RNN+kd	0.58	0.84	63.31	2.89	65.10	4.21	81.80	1.17	89.77	0.41
	RNN+pr	0.58	0.84	63.56	2.42	65.07	4.53	81.90	1.10	89.73	0.53
CRF	*FSTRNN*	74.87	0.06	76.08	0.50	75.92	0.47	82.92	0.72	90.07	0.18
	FSTGRU	74.87	0.06	75.85	0.50	75.92	0.47	83.48	0.67	90.73	0.51
	GRU	0.61	0.45	64.27	1.73	68.72	3.87	82.71	0.40	90.55	0.19
	GRU+i	0.28	0.49	64.20	1.92	71.43	2.94	83.39	0.64	90.45	0.84
	GRU+o	19.05	10.74	63.12	3.06	69.27	4.98	82.49	1.21	90.48	0.38
	GRU+io	20.11	12.81	64.23	2.92	71.62	2.63	82.65	0.81	90.91	0.76
	GRU+kd	0.61	0.45	62.12	3.25	68.72	3.87	82.52	0.09	90.52	1.24
	GRU+pr	0.61	0.45	62.46	2.39	68.72	3.87	82.71	0.40	90.75	0.42
	LSTM	0.92	1.07	63.28	1.50	66.26	3.72	82.83	0.59	90.94	0.49
	LSTM+i	0.11	0.08	63.05	2.35	70.88	2.24	83.49	0.15	89.54	0.64
	LSTM+o	20.73	11.42	63.51	1.16	67.22	3.38	82.19	1.56	90.44	0.41
	LSTM+io	22.06	13.23	63.97	2.10	70.53	3.45	83.59	1.04	90.68	0.08
	LSTM+kd	0.92	1.07	63.32	1.51	66.26	3.72	83.05	0.31	91.29	0.79
	LSTM+pr	0.92	1.07	63.28	1.50	66.26	3.72	83.10	0.24	90.43	0.92
	RNN	0.30	0.28	62.96	3.12	67.04	3.40	82.82	0.69	89.93	0.58
	RNN+i	0.71	0.91	65.75	1.87	71.40	2.84	82.68	0.74	89.59	0.64
	RNN+o	17.91	9.46	61.47	2.12	67.21	4.42	82.42	0.94	90.13	0.66
	RNN+io	14.80	12.25	65.47	2.02	71.82	1.83	82.06	0.93	89.96	0.10
	RNN+kd	0.30	0.28	62.92	3.16	67.04	3.16	82.84	0.91	89.71	0.31
	RNN+pr	0.30	0.28	62.96	3.12	67.04	3.40	82.84	0.91	89.64	0.17

Table 12: Full ATIS-ZH results