Marrying Up Regular Expressions with Neural Networks: A Case Study for Spoken Language Understanding

Bingfeng Luo[1], **Yansong Feng**[*1], **Zheng Wang**[2],
Songfang Huang[3], **Rui Yan**[1] and **Dongyan Zhao**[1]
[1]ICST, Peking University, China
[2]MetaLab, Lancaster University, UK
[3]IBM China Research Lab, China
{bf_luo,fengyansong,ruiyan,zhaody}@pku.edu.cn
z.wang@lancaster.ac.uk, huangsf@cn.ibm.com

Abstract

The success of many natural language processing (NLP) tasks is bound by the number and quality of annotated data, but there is often a shortage of such training data. In this paper, we ask the question: "Can we combine a neural network (NN) with regular expressions (RE) to improve supervised learning for NLP?". In answer, we develop novel methods to exploit the rich expressiveness of REs at different levels within a NN, showing that the combination significantly enhances the learning effectiveness when a small number of training examples are available. We evaluate our approach by applying it to spoken language understanding for intent detection and slot filling. Experimental results show that our approach is highly effective in exploiting the available training data, giving a clear boost to the RE-unaware NN.

1 Introduction

Regular expressions (REs) are widely used in various natural language processing (NLP) tasks like pattern matching, sentence classification, sequence labeling, etc. (Chang and Manning, 2014). As a technique based on human-crafted rules, it is concise, interpretable, tunable, and does not rely on much training data to generate. As such, it is commonly used in industry, especially when the available training examples are limited – a problem known as few-shot learning (GC et al., 2015).

While powerful, REs have a poor generalization ability because all synonyms and variations in a RE must be explicitly specified. As a result, REs are often ensembled with data-driven methods, such as neural network (NN) based techniques, where a set of carefully-written REs are used to handle certain cases with high precision, leaving the rest for data-driven methods.

We believe the use of REs can go beyond simple pattern matching. In addition to being a separate classifier to be ensembled, a RE also encodes a developer's knowledge for the problem domain. The knowledge could be, for example, the informative words (***clue words***) within a RE's surface form. We argue that such information can be utilized by data-driven methods to achieve better prediction results, especially in few-shot learning.

This work investigates the use of REs to improve NNs – a learning framework that is widely used in many NLP tasks (Goldberg, 2017). The combination of REs and a NN allows us to exploit the conciseness and effectiveness of REs and the strong generalization ability of NNs. This also provides us an opportunity to learn from various kinds of REs, since NNs are known to be good at tolerating noises (Xie et al., 2016).

This paper presents novel approaches to combine REs with a NN at different levels. At the input layer, we propose to use the evaluation outcome of REs as the input features of a NN (Sec.3.2). At the network module level, we show how to exploit the knowledge encoded in REs to guide the attention mechanism of a NN (Sec. 3.3). At the output layer, we combine the evaluation outcome of a RE with the NN output in a learnable manner (Sec. 3.4).

We evaluate our approach by applying it to two spoken language understanding (SLU) tasks, namely *intent detection* and *slot filling*, which respectively correspond to two fundamental NLP tasks: sentence classification and sequence labeling. To demonstrate the usefulness of REs in real-world scenarios where the available number of annotated data can vary, we explore both the few-shot learning setting and the one with full training data. Experimental results show that our approach is highly effective in utilizing the available

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 2083–2093
Melbourne, Australia, July 15 - 20, 2018.

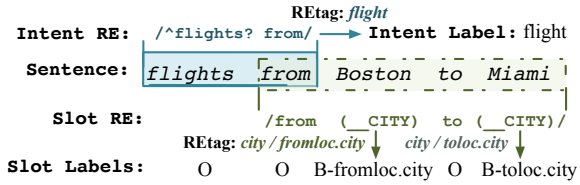


Figure 1: A sentence from the ATIS dataset. REs can be used to detect the intent and label slots.

annotated data, yielding significantly better learning performance over the RE-unaware method.

Our contributions are as follows. (1) We present the first work to systematically investigate methods for combining REs with NNs. (2) The proposed methods are shown to clearly improve the NN performance in both the few-shot learning and the full annotation settings. (3) We provide a set of guidance on how to combine REs with NNs and RE annotation.

2 Background

2.1 Typesetting

In this paper, we use italic for emphasis like *intent detection*, the Courier typeface for abbreviations like RE, bold italic for the first appearance of a concept like ***clue words***, Courier surrounded by / for regular expressions like `/list(the)?__AIRLINE/`, and underlined italic for words of sentences in our dataset like *Boston*.

2.2 Problem Definition

Our work targets two SLU tasks: *intent detection* and *slot filling*. The former is a sentence classification task where we learn a function to map an input sentence of n words, $\mathbf{x} = [x_1, ..., x_n]$, to a corresponding ***intent label***, c. The latter is a sequence labeling task for which we learn a function to take in an input query sentence of n words, $\mathbf{x} = [x_1, ..., x_n]$, to produce a corresponding labeling sequence, $\mathbf{y} = [y_1, ..., y_n]$, where y_i is the ***slot label*** of the corresponding word, x_i.

Take the sentence in Fig. 1 as an example. A successful intent detector would suggest the intent of the sentence as *flight*, i.e., querying about flight-related information. A slot filler, on the other hand, should identify the slots *fromloc.city* and *toloc.city* by labeling *Boston* and *Miami*, respectively, using the begin-inside-outside (BIO) scheme.

2.3 The Use of Regular Expressions

In this work, a RE defines a mapping from a text *pattern* to several ***REtags*** which are the same as or related to the ***target labels*** (i.e., intent and slot labels). A search function takes in a RE, applies it to all sentences, and returns any texts that match the pattern. We then assign the REtag(s) (that are associated with the matching RE) to either the matched sentence (for intent detection) or some matched phrases (for slot filling).

Specifically, our REtags for intent detection are the same as the intent labels. For example, in Fig. 1, we get a REtag of *flight* that is the same as the intent label *flight*.

For slot filling, we use two different sets of REs. Given the group functionality of RE, we can assign REtags to our interested ***RE groups*** (i.e., the expressions defined inside parentheses). The translation from REtags to slot labels depends on how the corresponding REs are used. (1) When REs are used at the network module level (Sec. 3.3), the corresponding REtags are the same as the target slot labels. For instance, the slot RE in Fig. 1 will assign *fromloc.city* to the first RE group and *toloc.city* to the second one. Here, __CITY is a list of city names, which can be replaced with a RE string like `/Boston|Miami|LA|.../`. (2) If REs are used in the input (Sec. 3.2) and the output layers (Sec. 3.4) of a NN, the corresponding REtag would be different from the target slot labels. In this context, the two RE groups in Fig. 1 would be simply tagged as *city* to capture the commonality of three related target slot labels: *fromloc.city*, *toloc.city*, *stoploc.city*. Note that we could use the target slot labels as REtags for all the settings. The purpose of abstracting REtags to a simplified version of the target slot labels here is to show that REs can still be useful when their evaluation outcome does not exactly match our learning objective. Further, as shown in Sec. 4.2, using simplified REtags can also make the development of REs easier in our tasks.

Intuitively, complicated REs can lead to better performance but require more efforts to generate. Generally, there are two aspects affecting RE complexity most: the number of RE groups[1] and *or* clauses (i.e., expressions separated by the disjunction operator |) in a RE group. Having a larger number of RE groups often leads to better

[1] When discussing complexity, we consider each semantically independent consecutive word sequence as a RE group (excluding clauses, such as \w+, that can match any word). For instance, the RE: `/how long(\w+){1,2}? (it take|flight)/` has two RE groups: `(how long)` and `(it take|flight)`.

precision but lower coverage on pattern matching, while a larger number of *or* clauses usually gives a higher coverage but slightly lower precision.

3 Our Approach

As depicted in Fig. 2, we propose to combine NNs and REs from three different angles.

3.1 Base Models

We use the Bi-directional LSTM (BLSTM) as our base NN model because it is effective in both intent detection and slot filling (Liu and Lane, 2016).

Intent Detection. As shown in Fig. 2, the BLSTM takes as input the word embeddings $[\mathbf{x}_1, ..., \mathbf{x}_n]$ of a n-word sentence, and produces a vector $\mathbf{h}_i$ for each word i. A self-attention layer then takes in the vectors produced by the BLSTM to compute the sentence embedding $\mathbf{s}$:

$$\mathbf{s} = \sum_i \alpha_i \mathbf{h}_i, \quad \alpha_i = \frac{\exp(\mathbf{h}_i^\intercal \mathbf{W} \mathbf{c})}{\sum_i \exp(\mathbf{h}_i^\intercal \mathbf{W} \mathbf{c})} \tag{1}$$

where α_i is the attention for word i, $\mathbf{c}$ is a randomly initialized trainable vector used to select informative words for classification, and $\mathbf{W}$ is a weight matrix. Finally, $\mathbf{s}$ is fed to a softmax classifier for intent classification.

Slot Filling. The model for slot filling is straightforward – the slot label prediction is generated by a softmax classier which takes in the BLSTM's output $\mathbf{h}_i$ and produces the slot label of word i. Note that attention aggregation in Fig. 2 is only employed by the network module level method presented in Sec. 3.3.

3.2 Using REs at the Input Level

At the input level, we use the evaluation outcomes of REs as features which are fed to NN models.

Intent Detection. Our REtag for intent detection is the same as our target intent label. Because real-world REs are unlikely to be perfect, one sentence may be matched by more than one RE. This may result in several REtags that are conflict with each other. For instance, the sentence *list the Delta airlines flights to Miami* can match a RE: `/list(the)?_AIRLINE/` that outputs tag *airline*, and another RE: `/list(\w+){0,3}flights?/` that outputs tag *flight*.

To resolve the conflicting situations illustrated above, we average the randomly initialized trainable tag embeddings to form an aggregated embedding as the NN input. There are two ways to use the aggregated embedding. We can append the aggregated embedding to either the embedding of every input word, or the input of the softmax classifier (see ① in Fig. 2(a)). To determine which strategy works best, we perform a pilot study. We found that the first method causes the tag embedding to be copied many times; consequently, the NN tends to heavily rely on the REtags, and the resulting performance is similar to the one given by using REs alone in few-shot settings. Thus, we adopt the second approach.

Slot Filling. Since the evaluation outcomes of slot REs are word-level tags, we can simply embed and average the REtags into a vector $\mathbf{f}_i$ for each word, and append it to the corresponding word embedding $\mathbf{w}_i$ (as shown in ① in Fig. 2(b)). Note that we also extend the slot REtags into the BIO format, e.g., the REtags of phrase *New York* are *B-city* and *I-city* if its original tag is *city*.

3.3 Using REs at the Network Module Level

At the network module level, we explore ways to utilize the clue words in the surface form of a RE (bold blue arrows and words in ② of Fig. 2) to guide the attention module in NNs.

Intent Detection. Taking the sentence in Fig. 1 for example, the RE: `/^flights?from/` that leads to intent *flight* means that *flights from* are the key words to decide the intent *flight*. Therefore, the attention module in NNs should leverage these two words to get the correct prediction. To this end, we extend the base intent model by making two changes to incorporate the guidance from REs.

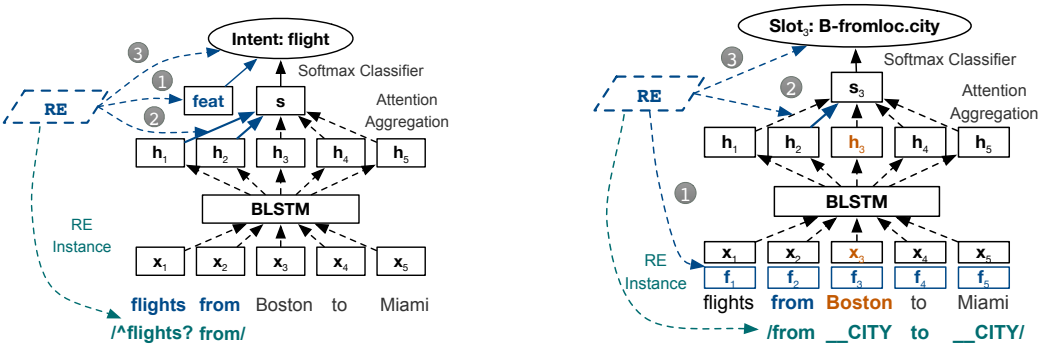
First, since each intent has its own clue words, using a single sentence embedding for all intent labels would make the attention less focused. Therefore, we let each intent label k use different attention $\mathbf{a}_k$, which is then used to generate the sentence embedding $\mathbf{s}_k$ for that intent:

$$\mathbf{s}_k = \sum_i \alpha_{ki} \mathbf{h}_i, \quad \alpha_{ki} = \frac{\exp(\mathbf{h}_i^\intercal \mathbf{W}_a \mathbf{c}_k)}{\sum_i \exp(\mathbf{h}_i^\intercal \mathbf{W}_a \mathbf{c}_k)} \tag{2}$$

where $\mathbf{c}_k$ is a trainable vector for intent k which is used to compute attention $\mathbf{a}_k$, $\mathbf{h}_i$ is the BLSTM output for word i, and $\mathbf{W}_a$ is a weight matrix.

The probability p_k that the input sentence expresses intent k is computed by:

$$p_k = \frac{\exp(logit_k)}{\sum_k \exp(logit_k)}, \qquad logit_k = \mathbf{w}_k \mathbf{s}_k + b_k \tag{3}$$



(a) Intent Detection

(b) Slot Filling (predicting slot label for *Boston*)

Figure 2: Overview of our methods. ①, ②, ③ refers to the methods in Sec. 3.2, 3.3, 3.4 respectively.

where $\mathbf{w}_k$, $logit_k$, b_k are weight vector, logit, and bias for intent k, respectively.

Second, apart from indicating a sentence for intent k (***positive REs***), a RE can also indicate that a sentence does not express intent k (***negative REs***). We thus use a new set of attention (***negative attentions***, in contrast to ***positive attentions***), to compute another set of logits for each intent with Eqs. 2 and 3. We denote the logits computed by positive attentions as $logit_{pk}$, and those by negative attentions as $logit_{nk}$, the final logit for intent k can then be calculated as:

$$logit_k = logit_{pk} - logit_{nk} \tag{4}$$

To use REs to guide attention, we add an attention loss to the final loss:

$$loss_{att} = \sum_k \sum_i t_{ki} \log(\alpha_{ki}) \tag{5}$$

where t_{ki} is set to 0 when none of the matched REs (that leads to intent k) marks word i as a clue word – otherwise t_{ki} is set to $1/l_k$, where l_k is the number of clue words for intent k (if no matched RE leads to intent k, then $t_{k*} = 0$). We use Eq. 5 to compute the positive attention loss, $loss_{att\_p}$, for positive REs and negative attention loss, $loss_{att\_n}$, for negative ones. The final loss is computed as:

$$loss = loss_c + \beta_p loss_{att\_p} + \beta_n loss_{att\_n} \tag{6}$$

where $loss_c$ is the original classification loss, β_p and β_n are weights for the two attention losses.

Slot Filling. The ***two-side attention*** (positive and negative attention) mechanism introduced for intent prediction is unsuitable for slot filling. Because for slot filling, we need to compute attention for each word, which demands more computational and memory resources than doing that for intent detection[2].

Because of the aforementioned reason, we use a simplified version of the two-side attention, where all the slot labels share the same set of positive and negative attention. Specifically, to predict the slot label of word i, we use the following equations, which are similar to Eq. 1, to generate a sentence embedding $\mathbf{s}_{pi}$ with regard to word i from positive attention:

$$\mathbf{s}_{pi} = \sum_j \alpha_{pij} \mathbf{h}_j, \quad \alpha_{pij} = \frac{\exp(\mathbf{h}_j^\intercal \mathbf{W}_{sp} \mathbf{h}_i)}{\sum_j \exp(\mathbf{h}_j^\intercal \mathbf{W}_{sp} \mathbf{h}_i)} \tag{7}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the BLSTM outputs for word i and j respectively, $\mathbf{W}_{sp}$ is a weight matrix, and α_{pij} is the positive attention value for word j with respect to word i. Further, by replacing $\mathbf{W}_{sp}$ with $\mathbf{W}_{sn}$, we use Eq. 7 again to compute negative attention and generate the corresponding sentence embedding $\mathbf{s}_{ni}$.

Finally, the prediction $\mathbf{p}_i$ for word i can be calculated as:

$$\begin{aligned} \mathbf{p}_i = \text{softmax}((\mathbf{W}_p[\mathbf{s}_{pi}; \mathbf{h}_i] + \mathbf{b}_p) \\ -(\mathbf{W}_n[\mathbf{s}_{ni}; \mathbf{h}_i] + \mathbf{b}_n)) \end{aligned} \tag{8}$$

where $\mathbf{W}_p$, $\mathbf{W}_n$, $\mathbf{b}_p$, $\mathbf{b}_n$ are weight matrices and bias vectors for positive and negative attention, respectively. Here we append the BLSTM output $\mathbf{h}_i$ to $\mathbf{s}_{pi}$ and $\mathbf{s}_{ni}$ because the word i itself also plays a crucial part in identifying its slot label.

3.4 Using REs at the Output Level

At the output level, REs are used to amend the output of NNs. At this level, we take the same

[2]Since we need to assign a label to each word, if we still compute attention for each slot label, we will have to compute $2 \times L \times n^2$ attention values for one sentence. Here, L is the number of tags and n is the sentence length. The BIO tagging format will further double the number of tags.

approach used for intent detection and slot filling (see ③ in Fig. 2).

As mentioned in Sec. 2.3, the slot REs used in the output level only produce a simplified version of target slot labels, for which we can further annotate their corresponding target slot labels. For instance, a RE that outputs *city* can lead to three slot labels: *fromloc.city*, *toloc.city*, *stoploc.city*.

Let z_k be a 0-1 indicator of whether there is at least one matched RE that leads to target label k (intent or slot label), the final logits of label k for a sentence (or a specific word for slot filling) is:

$$logit_k = logit'_k + w_k z_k \tag{9}$$

where $logit'_k$ is the logit produced by the original NN, and w_k is a trainable weight indicating the overall confidence for REs that lead to target label k. Here we do not assign a trainable weight for each RE because it is often that only a few sentences match a RE.

We modify the logit instead of the final probability because a logit is an unconstrained real value, which matches the property of $w_k z_k$ better than probability. Actually, when performing model ensemble, ensembling with logits is often empirically better than with the final probability[3]. This is also the reason why we choose to operate on logits in Sec. 3.3.

4 Evaluation Methodology

Our experiments aim to answer three questions: **Q1:** Does the use of REs enhance the learning quality when the number of annotated instances is small? **Q2:** Does the use of REs still help when using the full training data? **Q3:** How can we choose from different combination methods?

4.1 Datasets

We use the ATIS dataset (Hemphill et al., 1990) to evaluate our approach. This dataset is widely used in SLU research. It includes queries of flights, meal, etc. We follow the setup of Liu and Lane (2016) by using 4,978 queries for training and 893 for testing, with 18 intent labels and 127 slot labels. We also split words like *Miami's* into *Miami 's* during the tokenization phase to reduce the number of words that do not have a pre-trained word embedding. This strategy is useful for few-shot learning.

[3] An example can be found in the ensemble version that Juan et al. (2016) used in the Avazu Kaggle competition.

To answer **Q1** , we also exploit the ***full few-shot learning setting***. Specifically, for intent detection, we randomly select 5, 10, 20 training instances for each intent to form the few-shot training set; and for slot filling, we also explore 5, 10, 20 shots settings. However, since a sentence typically contains multiple slots, the number of mentions of frequent slot labels may inevitably exceeds the target shot count. To better approximate the target shot count, we select sentences for each slot label in ascending order of label frequencies. That is k_1-shot dataset will contain k_2-shot dataset if $k_1 > k_2$. All settings use the original test set.

Since most existing few-shot learning methods require either many few-shot classes or some classes with enough data for training, we also explore the ***partial few-shot learning setting*** for intent detection to provide a fair comparison for existing few-shot learning methods. Specifically, we let the 3 most frequent intents have 300 training instances, and the rest remains untouched. This is also a common scenario in real world, where we often have several frequent classes and many classes with limited data. As for slot filling, however, since the number of mentions of frequent slot labels already exceeds the target shot count, the original slot filling few-shot dataset can be directly used to train existing few-shot learning methods. Therefore, we do not distinguish full and partial few-shot learning for slot filling.

4.2 Preparing REs

We use the syntax of REs in Perl in this work. Our REs are written by a paid annotator who is familiar with the domain. It took the annotator in total less than 10 hours to develop all the REs, while a domain expert can accomplish the task faster. We use the 20-shot training data to develop the REs, but word lists like cities are obtained from the full training set. The development of REs is considered completed when the REs can cover most of the cases in the 20-shot training data with resonable precision. After that, the REs are fixed throughout the experiments.

The majority of the time for writing the REs is proportional to the number of RE groups. It took about 1.5 hours to write the 54 intent REs with on average 2.2 groups per RE. It is straightforward to write the slot REs for the input and output level methods, for which it took around 1 hour to write the 60 REs with 1.7 groups on average. By con-

trast, writing slot REs to guide attention requires more efforts as the annotator needs to carefully select clue words and annotate the full slot label. As a result, it took about 5.5 hours to generate 115 REs with on average 3.3 groups. The performance of the REs can be found in the last line of Table 1.

In practice, a positive RE for intent (or slot) k can often be treated as negative REs for other intents (or slots). As such, we use the positive REs for intent (or slot) k as the negative REs for other intents (or slots) in our experiments.

4.3 Experimental Setup

Hyper-parameters. Our hyper-parameters for the BLSTM are similar to the ones used by Liu and Lane (2016). Specifically, we use batch size 16, dropout probability 0.5, and BLSTM cell size 100. The attention loss weight is 16 (both positive and negative) for full few-shot learning settings and 1 for other settings. We use the 100d GloVe word vectors (Pennington et al., 2014) pre-trained on Wikipedia and Gigaword (Parker et al., 2011), and the Adam optimizer (Kingma and Ba, 2014) with learning rate 0.001.

Evaluation Metrics. We report accuracy and macro-F1 for intent detection, and micro/macro-F1 for slot filling. Micro/macro-F1 are the harmonic mean of micro/macro precision and recall. Macro-precision/recall are calculated by averaging precision/recall of each label, and micro-precision/recall are averaged over each prediction.

Competitors and Naming Conventions. Here, a bold Courier typeface like **BLSTM** denotes the notations of the models that we will compare in Sec. 5.

Specifically, we compare our methods with the baseline **BLSTM** model (Sec. 3.1). Since our attention loss method (Sec. 3.3) uses two-side attention, we include the raw two-side attention model without attention loss (**+two**) for comparison as well. Besides, we also evaluate the RE output (**REO**), which uses the REtags as prediction directly, to show the quality of the REs that we will use in the experiments.[4]

As for our methods for combinging REs with NN, **+feat** refers to using REtag as input features (Sec. 3.2), **+posi** and **+neg** refer to using positive and negative attention loss respectively, **+both** refers to using both postive and negative attention losses (Sec. 3.3), and **+logit** means using REtag to modify NN output (Sec. 3.4).

Moverover, since the REs can also be formatted as first-order-logic (FOL) rules, we also compare our methods with the teacher-student framework proposed by Hu et al. (2016a), which is a general framework for distilling knowledge from FOL rules into NN (**+hu16**). Besides, since we consider few-short learning, we also include the memory module proposed by Kaiser et al. (2017), which performs well in various few-shot datasets (**+mem**)[5]. Finally, the state-of-art model on the ATIS dataset is also included (**L&L16**), which jointly models the intent detection and slot filling in a single network (Liu and Lane, 2016).

5 Experimental Results

5.1 Full Few-Shot Learning

To answer **Q1** , we first explore the full few-shot learning scenario.

Intent Detection. As shown in Table 1, except for 5-shot, all approaches improve the baseline BLSTM. Our network-module-level methods give the best performance because our attention module directly receives signals from the clue words in REs that contain more meaningful information than the REtag itself used by other methods. We also observe that since negative REs are derived from positive REs with some noises, posi performs better than neg when the amount of available data is limited. However, neg is slightly better in 20-shot, possibly because negative REs significantly outnumbers the positive ones. Besides, two alone works better than the BLSTM when there are sufficient data, confirming the advantage of our two-side attention architecture.

As for other proposed methods, the output level method (logit) works generally better than the input level method (feat), except for the 5-shot case. We believe this is due to the fewer number of RE related parameters and the shorter distance that the gradient needs to travel from the loss to these parameters – both make logit easier to train. However, since logit directly modifies the output, the final prediction is more sensitive to the insufficiently trained weights in logit, leading to the inferior results in the 5-shot setting.

[4] For slot filling, we evaluate the REs that use the target slot labels as REtags.

[5] We tune C and π_0 of hu16, and choose (0.1, 0.3) for intent, and (1, 0.3) for slot. We tune memory-size and k of mem, and choose (1024, 64) for intent, and (2048, 64) for slot.

Model Type	Model Name	Intent			Slot		
		5-shot	10-shot	20-shot	5-shot	10-shot	20-shot
		Macro-F1 / Accuracy			Macro-F1 / Accuracy		
Base Model	BLSTM	45.28 / 60.02	60.62 / 64.61	63.60 / 80.52	60.78 / 83.91	74.28 / 90.19	80.57 / 93.08
Input Level	+feat	49.40 / 63.72	64.34 / 73.46	65.16 / 83.20	**66.84 / 88.96**	79.67 / **93.64**	84.95 / 95.00
Output Level	+logit	46.01 / 58.68	63.51 / 77.83	69.22 / **89.25**	63.68 / 86.18	76.12 / 91.64	83.71 / 94.43
	+hu16	47.22 / 56.22	61.83 / 68.42	67.40 / 84.10	63.37 / 85.37	75.67 / 91.06	80.85 / 93.47
Network Module Level	+two	40.44 / 57.22	60.72 / 75.14	62.88 / 83.65	60.38 / 83.63	73.22 / 90.08	79.58 / 92.57
	+two+posi	50.90 / 74.47	68.69 / 84.66	72.43 / 85.78	59.59 / 83.47	73.62 / 89.28	78.94 / 92.21
	+two+neg	49.01 / 68.31	64.67 / 79.17	72.32 / 86.34	59.51 / 83.23	72.92 / 89.11	78.83 / 92.07
	+two+both	**54.86 / 75.36**	**71.23 / 85.44**	**75.58** / 88.80	59.47 / 83.35	73.55 / 89.54	79.02 / 92.22
Few-Shot Model	+mem	-	-	-	61.25 / 83.45	77.83 / 90.57	82.98 / 93.49
	+mem+feat	-	-	-	65.08 / 88.07	**80.64** / 93.47	**85.45 / 95.39**
RE Output	REO	70.31 / 68.98			42.33 / 70.79		

Table 1: Results on Full Few-Shot Learning Settings. For slot filling, we do not distinguish full and partial few-shot learning settings (see Sec. 4.1).

To compare with existing methods of combining NN and rules, we also implement the teacher-student network (Hu et al., 2016a). This method lets the NN learn from the posterior label distribution produced by FOL rules in a teacher-student framework, but requires considerable amounts of data. Therefore, although both hu16 and logit operate at the output level, logit still performs better than hu16 in these few-shot settings, since logit is easier to train.

It can also be seen that starting from 10-shot, two+both significantly outperforms pure REO. This suggests that by using our attention loss to connect the distributional representation of the NN and the clue words of REs, we can generalize RE patterns within a NN architecture by using a small amount of annotated data.

Slot Filling. Different from intent detection, as shown in Table 1, our attention loss does not work for slot filling. The reason is that the slot label of a ***target word*** (the word for which we are trying to predict a slot label) is decided mainly by the semantic meaning of the word itself, together with 0-3 phrases in the context to provide supplementary information. However, our attention mechanism can only help in recognizing clue words in the context, which is less important than the word itself and have already been captured by the BLSTM, to some extent. Therefore, the attention loss and the attention related parameters are more of a burden than a benefit. As is shown in Fig. 1, the model recognizes *Boston* as *fromloc.city* mainly because *Boston* itself is a city, and its context word *from* may have already been captured by the BLSTM and our attention mechanism does not help much. By examining the attention values of +two trained on the full dataset, we find that instead of marking informative context words, the attention tends to concentrate on the target word itself. This observation further reinforces our hypothesis on the attention loss.

On the other hand, since the REtags provide extra information, such as type, about words in the sentence, logit and feat generally work better. However, different from intent detection, feat only outperforms logit by a margin. This is because feat can use the REtags of all words to generate better context representations through the NN, while logit can only utilize the REtag of the target word before the final output layer. As a result, feat actually gathers more information from REs and can make better use of them than logit. Again, hu16 is still outperformed by logit, possibly due to the insufficient data support in this few-shot scenario. We also see that even the BLSTM outperforms REO in 5-shot, indicating while it is hard to write high-quality RE patterns, using REs to boost NNs is still feasible.

Summary. The amount of extra information that a NN can utilize from the combined REs significantly affects the resulting performance. Thus, the attention loss methods work best for intent detection and feat works best for slot filling. We also see that the improvements from REs decreases as having more training data. This is not surprising because the implicit knowledge embedded in the REs are likely to have already been captured by a sufficient large annotated dataset and in this scenario using the REs will bring in fewer benefits.

5.2 Partial Few-Shot Learning

To better understand the relationship between our approach and existing few-shot learning methods, we also implement the memory network method

Model	5-shot	10-shot	20-shot
	Macro-F1 / Accuracy		
BLSTM	64.73 / 91.71	78.55 / 96.53	82.05 / 97.20
+hu16	65.22 / 91.94	84.49 / 96.75	84.80 / 97.42
+two	65.59 / 91.04	77.92 / 95.52	81.01 / 96.86
+two+both	66.62 / 92.05	85.75 / 96.98	**87.97 / 97.76**
+mem	67.54 / 91.83	82.16 / 96.75	84.69 / 97.42
+mem+posi	**70.46 / 93.06**	**86.03 / 97.09**	86.69 / 97.65

Table 2: Intent Detection Results on Partial Few-Shot Learning Setting.

Model	Intent	Slot
	Macro-F1/Accuracy	Macro-F1/Micro-F1
BLSTM	92.50 / 98.77	85.01 / 95.47
+feat	91.86 / 97.65	86.7 / 95.55
+logit	92.48 / 98.77	86.94 / 95.42
+hu16	93.09 / 98.77	85.74 / 95.33
+two	93.64 / 98.88	84.45 / 95.05
+two+both	**96.20 / 98.99**	85.44 / 95.27
+mem	93.42 / 98.77	85.72 / 95.37
+mem+posi/feat	94.36 / **98.99**	**87.82 / 95.90**
L&L16	- / 98.43	- / 95.98

Table 3: Results on Full Dataset. The left side of '/' applies for intent, and the right side for slot.

(Kaiser et al., 2017) which achieves good results in various few-shot datasets. We adapt their open-source code, and add their memory module (mem) to our BLSTM model.

Since the memory module requires to be trained on either many few-shot classes or several classes with extra data, we expand our full few-shot dataset for intent detection, so that the top 3 intent labels have 300 sentences (partial few-shot).

As shown in Table 2, mem works better than BLSTM, and our attention loss can be further combined with the memory module (mem+posi), with even better performance. hu16 also works here, but worse than two+both. Note that, the memory module requires the input sentence to have only one embedding, thus we only use one set of positive attention for combination.

As for slot filling, since we already have extra data for frequent tags in the original few-shot data (see Sec. 4.1), we use them directly to run the memory module. As shown in the bottom of Table 1, mem also improves the base BLSTM, and gains further boost when it is combined with feat[6].

5.3 Full Dataset

To answer **Q2**, we also evaluate our methods on the full dataset. As seen in Table 3, for intent detection, while two+both still works, feat and logit no longer give improvements. This shows

[6]For compactness, we only combine the best method in each task with mem, but others can also be combined.

Model	Intent		Slot	
	Macro-F1 / Accuracy		Macro-F1 / Micro-F1	
	Complex	Simple	Complex	Simple
BLSTM	63.60 / 80.52		80.57 / 93.08	
+feat	65.16/**83.20**	**66.51**/80.40	**84.95/95.00**	83.88/94.71
+logit	**69.22/89.25**	65.09/83.09	**83.71/94.43**	83.22/93.94
+both	**75.58/88.80**	74.51/87.46	-	-

Table 4: Results on 20-Shot Data with Simple REs. +both refers to +two +both for short.

that since both REtag and annotated data provide intent labels for the input sentence, the value of the extra noisy tag from RE become limited as we have more annotated data. However, as there is no guidance on attention in the annotations, the clue words from REs are still useful. Further, since feat concatenates REtags at the input level, the powerful NN makes it more likely to overfit than logit, therefore feat performs even worse when compared to the BLSTM.

As for slot filling, introducing feat and logit can still bring further improvements. This shows that the word type information contained in the REtags is still hard to be fully learned even when we have more annotated data. Moreover, different from few-shot settings, two+both has a better macro-F1 score than the BLSTM for this task, suggesting that better attention is still useful when the base model is properly trained.

Again, hu16 outperforms the BLSTM in both tasks, showing that although the REtags are noisy, their teacher-student network can still distill useful information. However, hu16 is a general framework to combine FOL rules, which is more indirect in transferring knowledge from rules to NN than our methods. Therefore, it is still inferior to attention loss in intent detection and feat in slot filling, which are designed to combine REs.

Further, mem generally works in this setting, and can receive further improvement by combining our fusion methods. We can also see that two+both works clearly better than the state-of-art method (L&L16) in intent detection, which jointly models the two tasks. And mem+feat is comparative to L&L16 in slot filling.

5.4 Impact of the RE Complexity

We now discuss how the RE complexity affects the performance of the combination. We choose to control the RE complexity by modifying the number of groups. Specifically, we reduce the number of groups for existing REs to decrease RE complexity. To mimic the process of writing simple

REs from scratch, we try our best to keep the key RE groups. For intent detection, all the REs are reduced to at most 2 groups. As for slot filling, we also reduce the REs to at most 2 groups, and for some simples case, we further reduce them into word-list patterns, e.g., (__CITY).

As shown in Table 4, the simple REs already deliver clear improvements to the base NN models, which shows the effectiveness of our methods, and indicates that simple REs are quite cost-efficient since these simple REs only contain 1-2 RE groups and thus very easy to produce. We can also see that using complex REs generally leads to better results compared to using simple REs. This indicates that when considering using REs to improve a NN model, we can start with simple REs, and gradually increase the RE complexity to improve the performance over time[7].

6 Related Work

Our work builds upon the following techniques, while qualitatively differing from each

NN with Rules. On the initialization side, Li et al. (2017) uses important n-grams to initialize the convolution filters. On the input side, Wang et al. (2017a) uses knowledge base rules to find relevant concepts for short texts to augment input. On the output side, Hu et al. (2016a; 2016b) and Guo et al. (2017) use FOL rules to rectify the output probability of NN, and then let NN learn from the rectified distribution in a teacher-student framework. Xiao et al. (2017), on the other hand, modifies the decoding score of NN by multiplying a weight derived from rules. On the loss function side, people modify the loss function to model the relationship between premise and conclusion (Demeester et al., 2016), and fit both human-annotated and rule-annotated labels (Alashkar et al., 2017). Since fusing in initialization or in loss function often require special properties of the task, these approaches are not applicable to our problem. Our work thus offers new ways to exploit RE rules at different levels of a NN.

NNs and REs. As for NNs and REs, previous work has tried to use RE to speed up the decoding phase of a NN (Strauß et al., 2016) and generating REs from natural language specifications of the RE (Locascio et al., 2016). By contrast, our work aims to use REs to improve the prediction ability of a NN.

Few-Shot Learning. Prior work either considers few-shot learning in a metric learning framework (Koch et al., 2015; Vinyals et al., 2016), or stores instances in a memory (Santoro et al., 2016; Kaiser et al., 2017) to match similar instances in the future. Wang et al. (2017b) further uses the semantic meaning of the class name itself to provide extra information for few-shot learning. Unlike these previous studies, we seek to use the human-generated REs to provide additional information.

Natural Language Understanding. Recurrent neural networks are proven to be effective in both intent detection (Ravuri and Stoicke, 2015) and slot filling (Mesnil et al., 2015). Researchers also find ways to jointly model the two tasks (Liu and Lane, 2016; Zhang and Wang, 2016). However, no work so far has combined REs and NNs to improve intent detection and slot filling.

7 Conclusions

In this paper, we investigate different ways to combine NNs and REs for solving typical SLU tasks. Our experiments demonstrate that the combination clearly improves the NN performance in both the few-shot learning and the full dataset settings. We show that by exploiting the implicit knowledge encoded within REs, one can significantly improve the learning performance. Specifically, we observe that using REs to guide the attention module works best for intent detection, and using REtags as features is an effective approach for slot filling. We provide interesting insights on how REs of various forms can be employed to improve NNs, showing that while simple REs are very cost-effective, complex REs generally yield better results.

Acknowledgement

This work is supported by the National High Technology R&D Program of China (Grant No. 2015AA015403), the National Natural Science Foundation of China (Grant Nos. 61672057 and 61672058); the UK Engineering and Physical Sciences Research Council (EPSRC) under grants EP/M01567X/1 (SANDeRs) and EP/M015793/1 (DIVIDEND); and the Royal Society International Collaboration Grant (IE161012). For any correspondence, please contact Yansong Feng.

[7] We do not include results of both for slot filling since its REs are different from feat and logit, and we have already shown that the attention loss method does not work for slot filling.

References

Taleb Alashkar, Songyao Jiang, Shuyang Wang, and Yun Fu. 2017. Examples-rules guided deep neural network for makeup recommendation. In *AAAI*, pages 941–947.

Angel X Chang and Christopher D Manning. 2014. Tokensregex: Defining cascaded regular expressions over tokens. *Tech. Rep. CSTR 2014-02*.

Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. 2016. Lifted rule injection for relation embeddings. *arXiv preprint arXiv:1606.08359*.

Paul Suganthan GC, Chong Sun, Haojun Zhang, Frank Yang, Narasimhan Rampalli, Shishir Prasad, Esteban Arcaute, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, et al. 2015. Why big data industrial systems need rules and what we can do about it. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 265–276. ACM.

Yoav Goldberg. 2017. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2017. Knowledge graph embedding with iterative guidance from soft rules. *arXiv preprint arXiv:1711.11231*.

Charles T Hemphill, John J Godfrey, George R Doddington, et al. 1990. The atis spoken language systems pilot corpus. In *Proceedings of the DARPA speech and natural language workshop*, pages 96–101.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016a. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*.

Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric P Xing. 2016b. Deep neural networks with massive learned knowledge. In *EMNLP*, pages 1670–1679.

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM.

Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. 2017. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*.

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

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2.

Shen Li, Zhe Zhao, Tao Liu, Renfen Hu, and Xiaoyong Du. 2017. Initializing convolutional filters with semantic features for text classification. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1885–1890.

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

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge. *arXiv preprint arXiv:1608.03000*.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(3):530–539.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition, linguistic data consortium. *Google Scholar*.

Jeffrey Pennington, Richard Socher, and Christopher 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.

Suman Ravuri and Andreas Stoicke. 2015. A comparative study of neural network models for lexical intent classification. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 368–374. IEEE.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850.

Tobias Strauß, Gundram Leifert, Tobias Grüning, and Roger Labahn. 2016. Regular expressions for decoding of neural network outputs. *Neural Networks*, 79:1–11.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638.

Jin Wang, Zhongyuan Wang, Dawei Zhang, and Jun Yan. 2017a. Combining knowledge with deep convolutional neural networks for short text classification. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2915–2921. AAAI Press.

Peng Wang, Lingqiao Liu, Chunhua Shen, Zi Huang, Anton van den Hengel, and Heng Tao Shen. 2017b. Multi-attention network for one shot learning. In

2017 IEEE conference on computer vision and pattern recognition, CVPR, pages 22–25.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2017. Symbolic priors for rnn-based semantic parsing. In *wenty-sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 4186–4192.

Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. 2016. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4753–4762.

Xiaodong Zhang and Houfeng Wang. 2016. A joint model of intent determination and slot filling for spoken language understanding. In *IJCAI*, pages 2993–2999.