Hierarchical Tagger with Multi-task Learning for Cross-domain Slot Filling

Xiao Wei¹, Yuke Si1,*, Shiquan Wang¹, Longbiao Wang¹,*, Jianwu Dang¹,²

¹Tianjin Key Laboratory of Cognitive Computing and Application, College of Intelligence and Computing, Tianjin University, Tianjin, China
²Japan Advanced Institute of Science and Technology, Ishikawa, Japan

{weixiao,siyuke,wangshiquan,longbiao_wang}@tju.edu.cn, jdang@jaist.ac.jp

Abstract

In task-oriented dialog systems, slot filling aims to identify the semantic slot type of each token in utterances. Due to the lack of sufficient supervised data in many scenarios, it is necessary to transfer knowledge by using cross-domain slot filling. Previous studies focus on building the relationships among similar slots across domains by providing additional descriptions, yet not fully utilizing prior information. In this study, we mainly make two novel improvements. First, we improve the hierarchical frameworks based on pre-trained models. For instance, we add domain descriptions to auxiliary information in the similarity layer to enhance the relationships. Second, we improve the independent fine-tuning with multi-task learning by using an auxiliary network, where the domain detection task is deliberately set up corresponding to the domain descriptions. Additionally, we also adopt an adversarial regularization to avoid over-fitting. Experimental results on SNIPS dataset show that our model significantly outperforms the best baseline by 16.11%, 11.06% and 8.77%, respectively in settings of 0-shot, 20-shot and 50-shot in terms of micro F1, which demonstrates our model has better generalization ability, especially for domain-specific slots.

Index Terms: cross-domain slot filling, multi-task learning, representation learning, adversarial regularization

1. Introduction

As an essential module of human-computer interaction, natural language understanding in task-oriented dialog systems usually includes three tasks: domain detection, intent classification and slot filling [1], where slot filling aims to identify the predefined semantic slot type of each token in user utterances of certain domains. Recently, supervised approaches have made great achievements with massive amounts of labeled data [2, 3]. However, there are still many scenarios lacking such data in practice, and its collection is expensive and time-consuming [4]. And in actual production, some domains with slots unseen before will always appear after deploying dialogue systems [5]. Therefore, to cope with the data scarcity issue, one solution is to study domain adaptation approaches by using cross-domain slot filling, which aims to transfer knowledge from data-rich source domains into data-poor target domains.

Across domains there are usually domain-specific slots besides domain-shared slots, resulting in the domain adaptation problem. As for domain-specific slots, most models adopt transfer learning [6–8] to learn and transfer shared knowledge across domains by directly training in the source domains and evaluating in the target domains. As for domain-specific slots, previous studies focus on building the relationships of the same or similar slots across domains by additionally providing auxiliary information on slots. However, they did not fully utilize the prior information and the limited data. Earlier models usually used one-step frameworks, and introduced slot descriptions as auxiliary information to compare similarities and further predict labels [9, 10], with the hypothesis that similar slots would have similar descriptions; And slot example values were further considered [11, 12]. Due to the data sparsity issue, models with one-step frameworks are hard to accurately capture the spans of slot values [13], so later models prefer hierarchical frameworks with two steps proposed in [13, 14] (as shown in Fig.1). Based on them, some techniques to enhance the relationships were gradually applied such as contrastive learning [15] and syntactic structure encoders [16]. The two-step frameworks may suffer from the error propagation problem. And there were some recent studies using pre-trained models [5, 17], such as pre-trained named entity recognition (NER) models, ELMo [18] and BERT [19], and demonstrated the power of pre-trained models for general representation learning. A disadvantage of these methods is the insufficient fine-tuning, resulting in limited performances.

Given the above challenges and issues, we focus on directly learning general representations, in addition to enhancing the relationships among similar slots across domains. We propose a Hierarchical Tagger based on pre-trained BERT and fine-tune it with Multi-task Learning (HTML). Our work mainly includes the following two aspects:

- We improve the hierarchical frameworks based on pre-trained models. We change the commonly used architecture in step 1 to BERT-CRF [21], and improve the similarity layer in step 2 by adding domain descriptions to the auxiliary information.
- We improve the fine-tuning strategy with multi-task learning [22, 23]. We build an auxiliary network to independently fine-tune the pre-trained encoder with multi-task learning, where we deliberately set up a domain detection task correspondingly. Additionally, we also adopt an adversarial regularization [15, 24].

Figure 1: Hierarchical frameworks. With traditional frameworks models directly predict the specific slot labels with one step, while with hierarchical frameworks as shown, models first detect whether tokens are slot values or not in step 1, that is, segment labels [20], and further detect which slot types the tokens belong to in step 2, that is, slot labels.
2. HTML: hierarchical tagger with multi-task learning

In zero-shot setting, slot filling can be defined as follows: $S_t$ denotes a collection of pre-defined slot types in source domains (denoted by $D_s$), and $S_t$ denotes that in target domain (denoted by $D_t$). The data of one user utterance with $n$ tokens in $D_s$ is denoted by $X_s = \{x_1, x_2, ..., x_n\}$, with labels denoted by $Y_s = \{y_1, y_2, ..., y_n\}$, where $Y_s \subset S_s$. And $X_t$ denotes that in $D_s$ with labels denoted by $Y_t$, where $Y_t \subset S_t$. Given all $X_s$ with $Y_s$ in $D_s$, and all $X_t$ without $Y_t$ in $D_t$, the model uses the former to train, and the latter to evaluate.

2.1. Hierarchical tagger with pre-trained BERT

Recent development in natural language processing (NLP) shows the power of pre-trained models, which are first pretrained on a large-scale unsupervised corpus, and then need to be fine-tuned for only a few epochs to fit downstream tasks [18, 19]. As a popular transfer learning, the technique can help models learn universal character-based representations [25, 26].

Encouraged by that, we first use pre-trained BERT as our encoder. And to alleviate error propagation with hierarchical frameworks, we change the commonly used architecture BiLSTM-CRF [27–29] to BERT-CRF [21] in step 1. We keep CRF layer because it can automatically learn the constraints to ensure that the final prediction result is valid. We also improve the similarity layer by adding domain descriptions to the auxiliary information in step 2.

As Fig.2(b) shows, we use hierarchical frameworks in the main network, including a shared embedding layer and a shared encoder layer followed by two hierarchical task-specific decoders. The encoder is pre-trained BERT, and the decoders respectively consist of a CRF layer for segment labels and a similarity layer for further slot labels. In addition, as Fig.2(c) shows, the similarity layer includes a BiLSTM and a similarity comparison module [13], where the representations of the improved auxiliary information are used. Our auxiliary information consists of slot descriptions referring to [9], two example values referring to [12] and the additional domain descriptions.

We denote the data of one user utterance with $n$ tokens by $X = \{x_1, x_2, ..., x_n\}$ and the embedding layer by $E$. The workflow in the main network can be formulated as follows:

$$e = E(X), \quad h = \text{BERT}(e),$$

where $e = [e_1, e_2, ..., e_n]$ is the embedding of the tokens, and $h = [h_1, h_2, ..., h_n]$ is the hidden states output by the shared BERT encoder. For segment labels, we use BERT-CRF architecture to predict them:

$$p = \text{CRF}(h),$$

where $p = [p_1, p_2, ..., p_n]$ is the logits of the tokens. For further slot labels, if tokens are predicted as B or I label in the previous step, the corresponding hidden states denoted by $[h_1, h_2, ..., h_{d}]$ will be passed to the similarity layer. We first calculate the representations of each slot value by its hidden states:

$$r_k = \sum_{i=1}^{L} \text{BiLSTM}(h_i),$$

where $r_k$ denotes the representations of the $k^{th}$ slot value in the utterance and is obtained by summing the representations of its all tokens. As for the auxiliary information on slots, we calculate the representation vectors of each slot by summing the hidden states of its $N$ tokens, and then concatenate them to obtain the representation matrix $M_{aux}$ [12]:

$$r_{aux} = \sum_{j=1}^{N} \text{BERT}(E(t_j)),$$

where $r_{aux}$ denotes the representation vector of the auxiliary information for one slot and the $t_j$ denotes its $j^{th}$ token. $r_{aux} \subset R^{d_s}$ and $M_{aux} \subset R^{N \times d_s}$, where $d_s$ is the dimension of the representations and $n_s$ is the size of $S_s \cup S_t$. Finally, we compare the similarity between $r_k$ and $M_{aux}$ to predict further slot labels:

$$s_k = M_{aux} \cdot r_k,$$

where $s_k$ is the logits of the $k^{th}$ slot value.
2.2. Fine-tuning with multi-task learning

Compared with single-task learning (STL), multi-task learning (MTL) provides an efficient way to fully utilize the limited supervised data from several (related) tasks [22, 23]. And even though our main task is only slot filling, it is still possible to improve its performance. Meanwhile, MTL as an implicit regularizer can prevent models from over-fitting specific tasks and increase the generalization ability.

For better fine-tuning, we improve the training strategy with multi-task learning (MTL). As Fig.2(a) shows, we simply build an auxiliary network to independently fine-tune the encoder only, which consists of the pre-trained BERT and three temporary task-specific linear layers [30]. The linear layers are respectively for segment labels, slot labels and domain labels, corresponding to the additional domain descriptions in auxiliary information. By independently fine-tuning, we can master and improve the effect of fine-tuning, and with MTL our model can leverage limited supervised data efficiently.

When training with data in source domains, we first load the pre-train BERT in the auxiliary network and then fine-tune it for only a few epochs with MTL. We save the best checkpoint when fine-tuning, and after loading the well-fine-tuned BERT encoder in the main network, we train the two hierarchical task-specific decoders. Finally, the data in the target domains is used for evaluation.

2.3. Fine-tuning with adversarial regularization

Aggressive fine-tuning often causes over-fitting due to the limited available data and the high complexity of the pre-trained models [24]. And due to the misaligned schema across domains such as domain-specific slots [12], the models in cross-domain settings are more susceptible to noisy utterances in training data, and the impact of over-fitting is greater, which prevents models from generalizing to unseen slots nicely. By applying adversarial regularization (AR) strategy [15, 24], models can avoid over-fitting to data in the source domains with increasing robustness.

Therefore, we introduce an additional AR for tasks in the auxiliary network, except for domain detection. Specifically, for domain detection in fine-tuning stage, we only use cross-entropy loss as the objective to optimize. And for the others, we additionally apply KL divergence to make perturbations referring to [24]:

\[ \min_{\theta} F(\theta) = L(\theta) + \lambda \mathcal{R}_s(\theta), \]  \hspace{1cm} (6)
\[ \mathcal{R}_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|x_i - \tilde{x}_i\|_p \leq \epsilon} l_s(f(\tilde{x}_i; \theta), f(x_i; \theta)), \]  \hspace{1cm} (7)

where \( L(\theta) \) is the loss function, \( \mathcal{R}_s(\theta) \) is the perturbation, and \( \epsilon > 0 \) is a tuning parameter. \( l_s(\cdot; \cdot) \) is chosen as the symmetrized KL-divergence here.

3. Experiments

3.1. Experimental settings

3.1.1. Dataset

To evaluate our approach, we experiment with SNIPS [31], a personal voice assistant dataset consisting of 7 domains (intents) and 39 slots, some of which are domain-shared, and others are domain-specific. To test our framework, we take turns selecting one domain as the target domain and others as the source domains.

Following the settings in [13], for the zero-shot setting, the data in the source domains is used to train. And the data in the target domain is divided into two groups: the first 500 samples as validation sets and the rest as test sets. For the few-shot settings, we further transfer 20 (1%) and 50 (2.5%) samples from the target domain to training sets on the basis of the zero-shot setting.

3.1.2. Baselines

In our experiments, we compare our model with the following three notable baseline models in this setting:

1. Conceptual Tagger (CT) [9]: This method applied one-step frameworks and utilized slot descriptions for similarity comparison to improve the performance on domain-specific slots in the target domain.
2. Robust Zero-shot Tagger (RZT) [12]: Based on CT, this method utilized slot example values besides slot descriptions, and explored the effect of different numbers of example values for zero-shot slot filling.
3. Coarse-to-fine Approach (Coach) [13]: This method proposed two-step frameworks for segment labels and further slot labels with slot descriptions with data augmentation.

3.1.3. Implementation details

Following [19], all texts are tokenized with wordpieces [19] and cut to spans of no more than 128 tokens. In the auxiliary network, we load the bert-base-uncased model [19] provided by Google to fine-tune. We set the batch size to 32, the learning rate to 5e-5, the max epoch number to 5, and the optimizer to Adamax [32]. We also set the dropout rates for slot detection to 0.3 while the other two to 0.1. For AR, we set \( \lambda_s \) to 1 and \( \epsilon \) to 1e-5. In the main network, corresponding to the dimension of BERT base model, we set the hidden size of BiLSTM in the similarity layer to 384 with 1 layer. And referring to [13], we set the batch size to 32, use the Adam [32] with a learning rate of 5e-4, and set the early stop with patience 10. We use CRF loss for segment detection and cross-entropy loss for slot detection.

3.2. Main results and discussions

Quantitative Analysis: Our main experimental results are shown in Table 1, where 0/20/50-shot means the model is trained with 0/20/50 additional sample(s) from the target domain. Our model (HTML) significantly outperforms the best baseline model (Coach) by 16.11%, 11.06% and 8.77%, respectively in settings of 0-shot, 20-shot and 50-shot in terms of micro F1.

Based on the improved hierarchical frameworks and in step 1, we change the commonly used architecture BiLSTM-CRF to BERT-CRF with pre-trained models, which helps our model learn the general pattern of slot values and greatly alleviates the error propagation problem. In addition, we also try other architectures based on BERT and find that BERT-CRF is faster and outperforms others [33].

It is obvious that the improvement of our model differs a lot between the settings of zero-shot and few-shot. This is because our work mainly improves the learning of general knowledge and representations, in addition to enhancing the relationships, which are all more beneficial for the experiments with no sample in the target domain.

Analysis on Seen and Unseen Slots: To analyze the effectiveness of our model in the 0-shot setting, we further test the model on both seen and unseen parts in the target domain. Note that an
Table 1: Slot F1-scores on SNIPS for different target domains. 0-shot denotes zero-shot setting, while 20-shot and 50-shot denote few-shot on 20 (1%) and 50 (2.5%) samples respectively. Scores in each row represent the performance of the leftmost target domain, and bold numbers indicate the best performance in each setting. HTML denotes our proposed model.

<table>
<thead>
<tr>
<th>Training Setting</th>
<th>0 - shot</th>
<th>20 - shot</th>
<th>50 - shot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Model</td>
<td>CT</td>
<td>RZT</td>
</tr>
<tr>
<td>AddToPlaylist</td>
<td>38.82</td>
<td>42.77</td>
<td>50.90</td>
</tr>
<tr>
<td>BookRestaurant</td>
<td>27.54</td>
<td>30.68</td>
<td>34.01</td>
</tr>
<tr>
<td>GetWeather</td>
<td>46.45</td>
<td>50.28</td>
<td>50.47</td>
</tr>
<tr>
<td>PlayMusic</td>
<td>32.86</td>
<td>33.12</td>
<td>32.01</td>
</tr>
<tr>
<td>RateBook</td>
<td>14.54</td>
<td>16.43</td>
<td>22.06</td>
</tr>
<tr>
<td>SearchCreativeWork</td>
<td>39.79</td>
<td>44.45</td>
<td>46.65</td>
</tr>
<tr>
<td>FindScreeningEvent</td>
<td>13.83</td>
<td>12.25</td>
<td>25.63</td>
</tr>
<tr>
<td>Average F1</td>
<td>30.55</td>
<td>32.85</td>
<td>37.39</td>
</tr>
</tbody>
</table>

Table 2: Average slot F1-scores on SNIPS on unseen and seen parts. Note that we use the division in [13], where there are no unseen utterances in the PlayMusic domain and no unseen utterances in the SearchCreativeEvent domain.

<table>
<thead>
<tr>
<th>Target</th>
<th>0 sample</th>
<th>20 samples</th>
<th>50 samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samples</td>
<td>unseen</td>
<td>seen</td>
<td>unseen</td>
</tr>
<tr>
<td>CT</td>
<td>27.10</td>
<td>44.18</td>
<td>50.13</td>
</tr>
<tr>
<td>RZT</td>
<td>28.28</td>
<td>47.15</td>
<td>52.56</td>
</tr>
<tr>
<td>Coach</td>
<td>34.09</td>
<td>51.93</td>
<td>64.16</td>
</tr>
<tr>
<td>HTML</td>
<td>48.25</td>
<td>59.93</td>
<td>73.84</td>
</tr>
</tbody>
</table>

3.3. Ablation studies

To further explore the specific influences of MTL, AR, and the domain descriptions in auxiliary information (simply denoted by D), we conduct the ablation studies in the 0-shot setting. As shown in Table 3, the first part is about using the auxiliary network for fine-tuning with STL, MTL, and MTL + AR respectively. The second part is about using the main network to train the decoders based on the well-fine-tuned encoder, including MTL, MTL + AR, MTL + AR + D. Note that + D means using the improved auxiliary information with the domain descriptions in the main network.

Experimental results show that in the auxiliary network (when fine-tuning), our model learns more general knowledge from large amounts of unsupervised data by using pre-trained models, learns more general representations from limited supervised data with MTL, and further increases the generalization ability with AR. In the main network, our hierarchical decoders are trained based on the well-fine-tuned encoder with increasing performance. And corresponding to the domain detection task in the auxiliary network, adding domain descriptions in the similarity layer further improves the performance of our model, which suggests that domain descriptions, as high-level summaries of utterances in certain scenarios, are useful for identifying unseen slots.

We also experiment with 2TL + AR without D, whose average F1-scores is 52.29%, and its performance is second only to the experiment with MTL + AR + D. Here 2TL means that two tasks are used for fine-tuning in the auxiliary network, one for segment labels and the other for slot labels.

4. Conclusions and future work

In this study, we proposed a Hierarchical Tagger based on pre-trained BERT and fine-tuned it with Multi-task Learning (HTML). We improved the hierarchical frameworks by using BERT-CRF and domain descriptions. And we improved the independent fine-tuning by using multi-task learning with the corresponding domain detection task. In addition, adversarial regularization was adopted. Finally, the task-specific decoders in the main network were trained to make hierarchical predictions. Experimental results showed that our model significantly outperformed the best baseline by 16.11%, 11.06% and 8.77%, respectively in settings of 0-shot, 20-shot and 50-shot in terms of micro F1, which demonstrated our model had better generalization ability, especially for domain-specific slots. Later we will experiment with more targeted pre-trained models such as ToD-BERT [34], and optimize the setting of linear layers in the auxiliary network. In addition, we will also consider improving the learning about samples in the target domains.

5. Acknowledgements

This work was supported partially by the National Key RD Program of China under Grant 2018YFB1305200 and the National Natural Science Foundation of China under Grant 62176182.
6. References


