Concept Transfer Learning for Adaptive Language Understanding

Su Zhu and **Kai Yu** *
Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering
SpeechLab, Department of Computer Science and Engineering
Brain Science and Technology Research Center
Shanghai Jiao Tong University, Shanghai, China
{paul2204,kai.yu}@sjtu.edu.cn

Abstract

Concept definition is important in language understanding (LU) adaptation since literal definition difference can easily lead to data sparsity even if different data sets are actually semantically correlated. To address this issue, in this paper, a novel concept transfer learning approach is proposed. Here, substructures within literal concept definition are investigated to reveal the relationship between concepts. A hierarchical semantic representation for concepts is proposed, where a semantic slot is represented as a composition of *atomic concepts*. Based on this new hierarchical representation, transfer learning approaches are developed for adaptive LU. The approaches are applied to two tasks: value set mismatch and domain adaptation, and evaluated on two LU benchmarks: ATIS and DSTC 2&3. Thorough empirical studies validate both the efficiency and effectiveness of the proposed method. In particular, we achieve state-of-the-art performance (F_1-score 96.08%) on ATIS by only using lexicon features.

1 Introduction

The language understanding (LU) module is a key component of dialogue system (DS), parsing user's utterances into corresponding semantic concepts (or semantic slots [1]). For example, the utterance *"Show me flights from Boston to New York"* can be parsed into *(from_city=Boston, to_city=New York)* (Pieraccini et al., 1992). Typically, the LU is seen as a plain slot filling task.

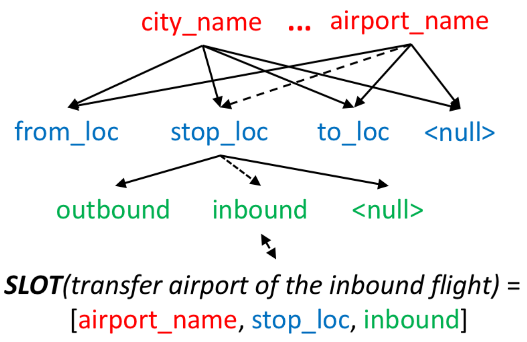

city_name
...
airport_name
from_loc
stop_loc
to_loc
<null>
outbound
inbound
<null>
SLOT(transfer airport of the inbound flight) =
[airport_name, stop_loc, inbound]

Figure 1: An example of hierarchical structure to represent semantic slot with atomic concepts. There are three levels in this structure. The plain slot $SLOT$(*transfer airport of the inbound flight*) can be represented as a tuple of atomic concepts sequentially.

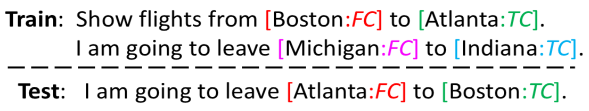
With sufficient in-domain data and deep learning models (e.g. recurrent neural networks, bidirectional long-short term memory network), statistical methods have achieved satisfactory performance in the slot filling task recently (Kurata et al., 2016; Vu, 2016; Liu and Lane, 2016).

However, retrieving sufficient in-domain data for training LU model (Tur et al., 2010) is unrealistic, especially when the semantic slot extends or dialogue domain changes. The ability of LU approaches to cope with changed domains and limited data is a key to the deployment of commercial dialogue systems (e.g. Apple Siri, Amazon Alexa, Google Home, Microsoft Cortana etc).

In this paper, we investigate substructure of semantic slots to find out slot relations and promote data reuse. We represent semantic slots with a hierarchical structure based on atomic concept tuple, as shown in Figure 1. Each semantic slot is composed of different atomic concepts, e.g. slot *"from_city"* can be defined as a tuple of atoms [*"from_location"*, *"city_name"*],

*The corresponding author is Kai Yu.

[1] Slot and concept are equal in LU. They will be mixed in the rest of this paper to some extent.


Train: Show flights from [Boston:*FC*] to [Atlanta:*TC*].
I am going to leave [Michigan:*FC*] to [Indiana:*TC*].

Test: I am going to leave [Atlanta:*FC*] to [Boston:*TC*].

Figure 2: An example of mismatched LU datasets labelled with *[value: slot]*. *FC* refers to *"from_city"*. *TC* refers to *"to_city"*.

and *"date_of_birth"* can be defined as [*"date"*, *"birth"*].

Unlike the traditional slot definition on a plain level, modeling on the atomic concepts helps identify linguistic patterns of related slots by atom sharing, and even decrease the required amount of training data. For example, the training and test sets are unmatched in Figure 2, whereas the patterns of atomic concepts (e.g. *"from"*, *"to"*, *"city"*) can be shared.

In this paper, we investigate the slot filling task switching from plain slots to hierarchical structures by proposing the novel atomic concept tuples which are constructed manually. For comparison, we also introduce a competitive method which automatically learns slot representation from the word sequence of each slot name. Our methods are applied to value set mismatch and domain adaptation problems on ATIS (Hemphill et al., 1995) and DSTC 2&3 (Henderson et al., 2013) respectively. As shown in the experimental results, the slot-filling based on concept transfer learning is effective in solving the value set mismatch and domain adaptation problems. The concept transfer learning method especially achieves state-of-the-art performance (F_1-score 96.08%) on the ATIS task.

The rest of the paper is organized as follows. The next section is about the relation to prior work. The atomic concept tuple is introduced in section 3. The proposed concept transfer learning is then described in section 4. Section 5 describes a competitive method with slot embedding derived from the literal descriptions of slot names. In section 6, the proposed approach is evaluated on the value set mismatch and domain adaptation problems. Finally, our conclusions are presented in section 7.

2 Related Work

Slot Filling in LU Zettlemoyer and Collins (2007) proposed a grammar induction method by learning a Probabilistic Combinatory Categorial Grammar (PCCG) from logical-form annotations. As a grammar-based method, PCCG is close to a hierarchical concepts structure in grammar generation and combination. But this grammar-based method does not possess high generalization capability for atomic concept sharing, and heavily depends on a well-defined lexicon set.

Recent research on statistical slot filling in LU has been focused on the Recurrent Neural Network (RNN) and its extensions. At first, RNN outperformed CRF (Conditional Random Field) on the ATIS dataset (Yao et al., 2013; Mesnil et al., 2013). Long-short term memory network (LSTM) was introduced to obtain a marginal improvement over RNN (Yao et al., 2014). After that, many RNN variations were proposed: encoder-labeler model (Kurata et al., 2016), attention model (Liu and Lane, 2016; Zhu and Yu, 2017) etc. However, these work only predicted the plain semantic slot, not the structure of atomic concepts.

Domain Adaptation in LU For the domain adaptation in LU, Zhu et al. (2014) proposed generating spoken language surface forms by using patterns of the source domain and the ontology of the target domain. With regard to the unsupervised LU, Heck and Hakkani-Tur (2012) exploited the structure of semantic knowledge graphs from the web to create natural language surface forms of entity-relation-entity portions of knowledge graphs. For the zero-shot learning of LU, Ferreira et al. (2015); Yazdani and Henderson (2015) proposed a model to calculate similarity scores between an input sentence and semantic items. In this paper, we focus on the extension of slots with limited seed data.

3 Atomic Concept Tuples

Although concept definition is one of the most crucial problems of LU, there is no unified surface form for the domain ontology. Even for the same semantic slot, names of this slot may be quite different. For example, the city where the flight departs may be called *"from_city"*, *"depart_city"* or *"from_loc.city_name"*. Ontology definitions from different groups may be similar but not consistent, which is not convenient for data reuse. Meanwhile, semantic slots defined in traditional LU systems are on a plain level, while there is no structure to indicate their relation.

To solve this problem, we propose to use atomic concepts to represent the semantic slots. Atomic concepts are exploited to break down the slots. We represent the semantic slots as atomic concept tuples (Figure 1 is an example). The semantic slot composed of these atomic concepts can keep a unified resource for concept definition and extend the semantic knowledge flexibly.

We propose a criteria to construct atomic concept manually. For a given vocabulary C of the atomic concepts, a semantic slot s can be represented by a tuple $[c_1, c_2, ..., c_k]$, where $c_i \in C$ is in the i-th dimension and k is tuple length. In particular, a *"null"* atom is introduced for each dimension. Table 1 illustrates an example of slot representation on the ATIS task. To avoid a scratch concept branch, we make a constraint:

$$C_i \cap C_j = \{null\}, 1 \leq i \neq j \leq k$$

where C_i $(1 \leq i \leq k)$ denotes all possible atomic concepts which exist in dimension i (i.e. $c_i \in C_i$). The concept tuple is ordered.

In general, atomic concepts can be classified into two categories, one is value-aware and the other is context-aware. The principle for defining slot as a concept branch is: lower dimension less context-aware. For example, *"city_name"* and *"airport_name"* depend on rare context (value-aware). They should be located in the first dimension. *"from_location"* depends on the context like a pattern of *"a flight leaves [city_name]"*, which should be in the second dimension. The atomic concept tuple shows the inner relation between different semantic slots explicitly.

slot	**atomic concept tuple**
city	[*city_name*, *null*]
from_city	[*city_name*, *from_location*]
depart_city	[*city_name*, *from_location*]
arrive_airport	[*airport_name*, *to_location*]

Table 1: An example of slot representation by atomic concepts.

Therefore, the procedure of constructing atomic concept tuples for slots can be divided into the following steps.

- Firstly, we build a vocabulary C of the atomic concepts for all the slots. By analyzing the conceptual intersection of different slots, we can split the slots into smaller ones which are called atomic concepts. After that, each slot is represented as a set of atomic concepts which are not ordered.
- Secondly, we gather the atoms into different groups. Atomic concepts from the same group should be mutually exclusive. Therefore we can investigate the inner relation and outer relation of these groups.
- Finally, each group is associated with one dimension (C_i) of the atomic concept tuple. The groups are ordered depending on whether they are value-aware or context-aware.

4 Concept Transfer Learning

The slot filling is typically considered as a sequence labelling problem. In this paper, we only consider the sequence-labelling based slot filling task. The input (word) sequence is denoted by $\mathbf{w} = (w_1, w_2, ..., w_N)$, and the output (slot tag) sequence is denoted by $\mathbf{s} = (s_1, s_2, ..., s_N)$. Since a slot may be mapped to several continuous words, we follow the popular in/out/begin (IOB) representation (e.g. an example in Figure 3).

Words	show	flights	from	Boston	to	New	York	today
Slots	O	O	O	B-FromCity	O	B-ToCity	I-ToCity	B-Date

Figure 3: An example of annotation for slot filling.

The typical slot filling task predicts a plain slot sequence given a word sequence, dubbed as **plain slot-filling** (**PS**).

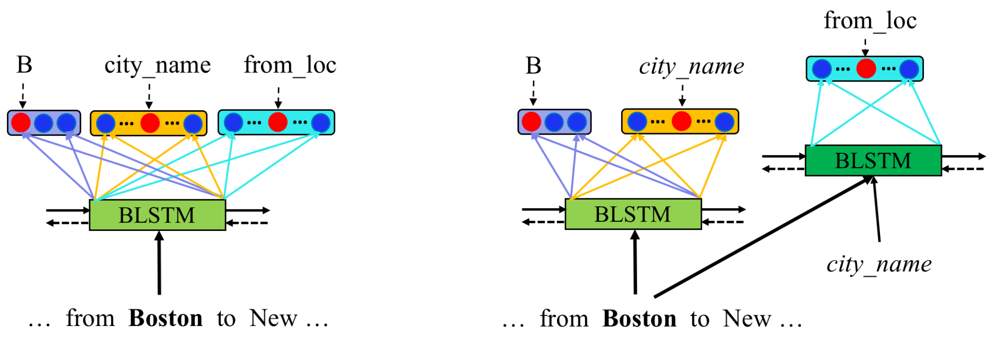
In this paper, the popular bidirectional LSTM-RNN (BLSTM) is used to model the sequence labeling problem (Graves, 2012). It can be exploited to capture both past and future features for a specific time frame. The BLSTM reads the input sentence $\mathbf{w}$ and generates N hidden states $h_i = \overleftarrow{h_i} \oplus \overrightarrow{h_i}, i \in \{1, .., N\}$:

$$\overleftarrow{h_i} = b(\overleftarrow{h_{i+1}}, e_{w_i}); \ \overrightarrow{h_i} = f(\overrightarrow{h_{i-1}}, e_{w_i})$$

where $\overleftarrow{h_i}$ is the hidden vector of the backward pass in BLSTM and $\overrightarrow{h_i}$ is the hidden vector of the forward pass in BLSTM at time i, b and f are LSTM units of the backward and forward passes respectively, e_w denotes the word embedding for each word w, and $\oplus$ denotes the vector concatenation operation. We write the entire operation as a mapping BLSTM$_{\Theta^w}$ (Θ^w refers to the parameters):

$$(h_1...h_N) = \text{BLSTM}_{\Theta^w}(w_1...w_N) \qquad (1)$$

Therefore, the plain slot filling defines a distribution over slot tag sequences given an input word



(a) *Atomic concept independent (**AC**)* (b) *Atomic concept dependent (**ACD**)*

Figure 4: The proposed method about the atomic-concepts based slot filling. A slot is considered as a tuple of atomic concepts, e.g. *"from_city"* is represented as [*"city_name"*, *"from_loc"*]. Multiple output layers are utilized to predict different atoms (including IOB schema). We involve two architectures: a) the AC assumes that the output layers are independent, b) while the ACD makes a dependence assumption.

sequence:

$$\begin{aligned} p(\mathbf{s}|\mathbf{w}) &= \prod_{i=1}^{N} p(s_i|h_i) \\ &= \prod_{i=1}^{N} \text{softmax}(W_o \cdot h_i)^T \delta_{s_i} \end{aligned} \tag{2}$$

where the matrix W_o (output layer) consists of the vector representations of each slot tag, the symbol δ_d is a Kronecker delta with a dimension for each slot tag, and the *softmax* function is used to estimate the probability distribution over all possible plain slots.

4.1 Atomic-Concepts Based Slot Filling

The slot is indicated as an atomic concept tuple based on hierarchical concept structure. Slot filling is considered as a concept-tuple labelling task.

(a) Atomic concept independent

Slot filling can be transferred to a multi-task sequence labelling problem, regarding these atomic concepts **independently** (i.e. **AC**). Each task predicts one atomic concept by a respective output layer. Thus, the slot filling problem can be formulated as

$$p(\mathbf{s}|\mathbf{w}) = \prod_{i=1}^{N} [p(\text{IOB}_i|h_i) \prod_{j=1}^{k} p(c_{ij}|h_i)]$$

where the semantic slot s_i is represented by an atomic concept branch $[c_{i1}, c_{i2}, ..., c_{ik}]$, and IOB_i is the IOB schema tag at time i. As illustrated in Figure 4(a), the semantic slot "*from_city*" can be represented as [*"city_name"*, *"from_loc"*]. The prediction of IOB is regarded as another task specifically. All tasks share the same parameters except for the output layers.

(b) Atomic concept dependent

Atomic concepts can also be regarded **dependently** (i.e. **ACD**) so that atomic concept prediction depends on the former predicted results. The slot filling problem can be formulated as

$$\begin{aligned} &p(\mathbf{s}|\mathbf{w}) \\ &= \prod_{i=1}^{N} [p(\text{IOB}_i|h_i) p(c_{i1}|h_i) \prod_{j=2}^{k} p(c_{ij}|h_i, c_{i,1:j-1})] \end{aligned}$$

where $c_{i,1:j-1} = (c_{i,1}, ..., c_{i,j-1})$ is the predicted result of former atomic concepts of slot tag s_i, indicating a structured multi-task learning framework.

In this paper, we make some simplifications on concept dependence. We predict atomic concept only based on the last atomic concept, as shown in Figure 4(b).

4.2 Training and Decoding

Since our approach is a structured multi-task learning problem, the model loss is summed over each task during training. For the domain adaptation, we firstly gather training data from the source domain and seed data from the target domain to be a union set. Subsequently, the union data is fed into the slot filling model.

During the decoding stage, we combine predicted atomic concepts with probability multiplication. The evaluation is made on the top-best hypothesis. Although the atomic-concepts based slot filling may predict an unseen slot. We didn't perform any post-processing but considered the unseen slot as a wrong prediction.

5 Literal Description of Slot Name

In the section, we introduce a competitive system which uses the literal description of the slot as an input of the slot filling model. The literal description of slot used in this paper is the word sequence of each slot name, which can be obtained automatically. As the names of relative slots may include the same or similar word, the word sequence of slot name can also help reveal the relation between different slots. Therefore, it is very meaningful to compare this method with the atomic concept tuples involving human knowledge.

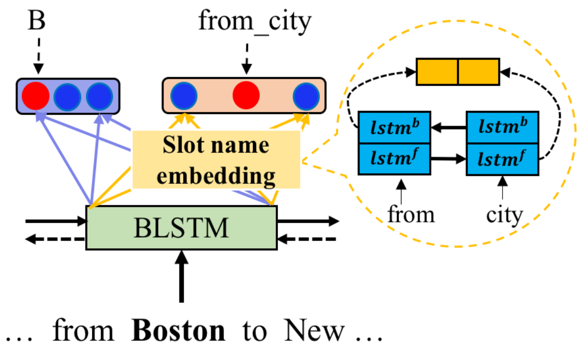


Figure 5: The proposed framework of slot filling based on the literal description of the slot. The literal description of a slot is the word sequence of slot name which can be obtained automatically, e.g. *"from_city"* is represented as a word sequence of "from city". Another BLSTM in the orange dotted circle is exploited to derive softmax embeddings from the slot names.

The architecture of this competitive system is illustrated in Figure 5. First, it assumes that each slot name is a meaningful natural language description so that the slot filling task is tractable from the input word sequence and slot name. Second, another BLSTM model is applied to derive softmax embedding from the slot names. In this method, we also split the slot filling task into IOB tag prediction and slot name prediction. In other words, the slot tag s_i is broken down into IOB$_i$ and slot name SN$_i$, e.g. the slot tag "B-*from_city*" is split into "B" and "*from_city*". The details are indicated below.

With the BLSTM applied on the input sequence, we have hidden vectors $h_i, i \in \{1, .., N\}$ as shown in Eqn. (1). This model redefines the distribution over slot tag sequences given an input word sequence, compared with Eqn. (2):

$$p(\mathbf{s}|\mathbf{w}) = \prod_{i=1}^{N} p(\text{IOB}_i|h_i)p(\text{SN}_i|h_i)$$

where $p(\text{IOB}_i|h_i)$ predicts the IOB tag and $p(\text{SN}_i|h_i)$ makes a prediction for the slot name. We define

$$p(\text{SN}_i|h_i) = \text{softmax}(W \cdot h_i)^T \delta_{\text{SN}_i}$$

where $W \in \mathbb{R}^{A \times B}$ is a matrix, $h_i \in \mathbb{R}^B$ is a vector, A is the number of all different slot names. The matrix W consists of the embedding of each slot name (i.e. each row vector of W with length B).

To capture the slot relation within different slot names, we apply another BLSTM model (as shown in the orange dotted circle of Figure 5) onto the word sequence (literal description) of each slot name. For the j-th slot name ($j \in \{1, .., A\}$) with a word sequence $\mathbf{x}^j = (x_1^j, ..., x_{N_j}^j)$, we have

$$\overleftarrow{v_n^j} = \text{lstm}^b(\overleftarrow{v_{n+1}^j}, e_{x_n^j}); \ \overrightarrow{v_n^j} = \text{lstm}^f(\overrightarrow{v_{n-1}^j}, e_{x_n^j})$$

where $\overleftarrow{v_n^j}$ is the hidden vector of the backward pass and $\overrightarrow{v_n^j}$ is the hidden vector of the forward pass at time n ($n \in \{1, .., N_j\}$), e_x denotes the word embedding for each word x. We take the tails of both backward and forward pass as the slot embedding, i.e.

$$W_j = \overleftarrow{v_1^j} \oplus \overrightarrow{v_{N_j}^j}$$

where W_j is the j-th row vector of matrix W.

The relative slots using the same or similar word in slot naming will be close in the space of slot embedding inherently. Therefore, this method is a competitive system to the atomic concept tuples. We will show the comparison in the following section.

6 Experiments

We evaluate our atomic-concept methods on two tasks: value set mismatch and domain adaptation.

Value set mismatch task evaluates the generalization capability of different slot filling models. In a language understanding (LU) system, each slot has a value set with all possible values which can be assigned to it. Since the semantically annotated data is always limited, only a part of values is seen in the training data. Will the slot filling model perform well on the unseen values? To answer this question, we synthesize a test set by the values mismatched with the training set of ATIS corpus. Our methods may take advantages of the prior knowledge about slot relations based on the atomic concepts and the literal descriptions of slot names.

Domain adaptation task evaluates the adaptation capability of our methods when they meet new slots in the target domain. In this task, a seed training set of the target domain is provided. However, it is very limited: 1) some new slots may not be covered; 2) not all contexts are covered for each new slot. The atomic-concepts based method would alleviate this problem. Each slot is defined as a tuple of atomic concepts in our method. Therefore, it is possible to learn an unseen slot of the target domain if its atomic concepts exist in the data of the source domain and the seed data of the target domain. It is also possible to see more contexts for a new slot if its atomic concepts exist in the source domain which has much more data.

6.1 Value Set Mismatch

ATIS corpus has been widely used as a benchmark by the LU community. The training data consists of 4978 sentences and the test data consists of 893 sentences.

In this task, we perform an adaptation for unmatched training and test sets, in which there are many unseen slot-value pairs in the test set (Figure 2 is an example). It is a common problem in the development of commercial dialogue system since it is impossible to collect data covering all possible slot-value pairs. We simulate this problem on the ATIS dataset (Hemphill et al., 1995) by creating an unmatched test set (**ATIS_X_test**).

ATIS_X_test is synthesized from the standard ATIS test set by randomly replacing the value of each slot with an unseen one. The unseen value sets are collected from the training set according to bottom-level concepts (e.g. *"city_name"*, *"airport_name"*). For example, if the value set of *"from_city"* is {*"New York"*, *"Boston"*} and the value set of *"to_city"* is {*"Boston"*}, then the unseen value for *"to_city"* is *"New York"*. The test sentence *"Flights to [xx:to_city]"* can be replaced to *"Flights to [New York:to_city]"*. Finally, the ATIS_X_test gets the same sentence number to the standard ATIS test set.

6.1.1 Experimental Settings

We randomly selected 80% of the training data for model training and the remaining 20% for validation. We deal with unseen words in the test set by marking any words with only one single occurrence in the training set as $\langle unk \rangle$. We also converted sequences of numbers to the string DIGIT, e.g. 1990 is converted to DIGIT*4 (Zhang and Wang, 2016). Regarding BLSTM model, we set the dimension of word embeddings to 100 and the number of hidden units to 100. For training, the network parameters are randomly initialized in accordance with the uniform distribution (-0.2, 0.2). Stochastic gradient descent (SGD) is used for updating parameters. The *dropout* with a probability of 0.5 is applied to the non-recurrent connections during the training stage.

We try different learning rates by grid-search in range of $[0.008, 0.04]$. We keep the learning rate for 100 epochs and save the parameters that give the best performance on the validation set. Finally, we report the F_1-score of the semantic slots on the test set with parameters that have achieved the best F_1-score on the validation set. The F_1-score is calculated using CoNLL evaluation script. [2]

6.1.2 Experimental Results and Analysis

Table 2 summarizes the recently published results on the ATIS slot filling task and compares them with the results of our proposed methods on the standard ATIS test set. We can see that RNN outperforms CRF because of the ability to capture long-term dependencies. LSTM beats RNN by solving the problem of vanishing or exploding gradients. BLSTM further improves the result by considering both the past and future features. Encoder-decoder achieves the state-of-the-art performance by modeling the label dependencies. Encoder-labeler is a similar method to the Encoder-decoder. These systems are designed to predict the plain semantic slots traditionally.

Compared with the published results, our method outperforms the previously published F1-score, illustrated in Table 2. **AC** gets a marginal improvement (+0.15%) over **PS** by predicting the atomic concepts independently instead of the plain slots. Moreover, **ACD** predicts the atomic concepts dependently, gains 0.50% (significant level 95%) over the **AC**. Worth to mention that **ACD** achieves a new state-of-the-art performance of the

[2]http://www.cnts.ua.ac.be/conll2000/chunking/output.html

Model	ATIS	ATIS_X_test
CRF (Mesnil et al., 2013)	92.94	–
RNN (Mesnil et al., 2013)	94.11	–
LSTM (Yao et al., 2014)	94.85	–
BLSTM (Zhang and Wang, 2016)	95.14	–
Encoder-decoder (Liu and Lane, 2016)	95.72	–
Encoder-labeler (Kurata et al., 2016)	95.66	–
Encoder-decoder-pointer (Zhai et al., 2017)	95.86	–
Encoder-decoder*	95.79	79.84
BLSTM* (PS)	95.43	79.59
PS + dict-feats	95.57	80.74
AC	95.58	80.90
ACD	**96.08**	**86.16**
Slot name embedding	95.52	81.49

Table 2: Comparison with the published results on the standard ATIS task, and evaluation on ATIS_X_test. ($*$ denotes our implementation.)

standard slot-tagging task on the ATIS dataset, with only the lexicon features [3].

Our methods are also tested on the ATIS_X_test to measure the ability of generalization. For comparison, we also apply dictionary features (n-gram indication) of value sets (e.g. some kind of gazetteers) collected from training data into the **PS** model (i.e. PS+*dict-feats* in Table 2). From Table 2, we can see that: 1) The plain slot filling models (**PS**, Encoder-decoder) are not on par with other models. 2) The atomic-concepts based slot filling gets a slight improvement over the **PS** with *dict-feats*, considering the concepts independently (**AC**). 3) The atomic-concepts based slot fillings (**ACD** gains a large margin over **AC**, considering the concepts dependently. 4) The method based on slot name embedding (described in Section 5) achieves a slight improvement than **AC**, which implies that it is possible to reveal the relationship between slots automatically.

[3]There are other published results that achieved better performance by using Name Entity features, e.g. Mesnil et al. (2013) got 96.24% F_1-score. The NE features are manually annotated and strong information. So it would be more meaningful to use only lexicon features. Meanwhile, several other works can obtain competitive results by using the intent classification as another task for joint training, e.g. Liu and Lane (2016) achieved 95.98% F_1-score. In this paper, we consider the slot filling task only.

Case study: As illustrated in Table 3, the plain slot filling (PS) predicts the label of *"late"* wrongly, whereas the atomic-concepts based slot fillings (i.e. AC and ACD) get the accurate annotation. The word of "late" is never covered by the slot *"period_of_day"* in the training set. It is hard for the plain slot filling (PS) to predict an unseen mapping correctly. Luckily, the "late" is covered by the family of the slot *"period_of_day"* in the training set, e.g. *"arrive_time.period_of_day"*. Therefore, AC and ACD can learn this by modeling the atomic concepts separately.

6.2 Domain Adaptation

Our methods are also evaluated on the DSTC 2&3 task (Henderson et al., 2013) which is considered to be a realistic domain adaptation problem.

DSTC 2 (source domain) comprises of dialogues from the restaurant information domain in Cambridge. We use the **dstc2_train** set (1612 dialogues) for training and the **dstc2_dev** (506 dialogues) for validation.

DSTC 3 (target domain) introduces the tourist information domain about restaurant, pubs and coffee shops in Cambridge, which is an extension of DSTC 2. We use seed data **dstc3_seed** (only 11 dialogues) as the training set of the target domain.

DSTC3_S_test: In this paper, we focus on three new semantic slots: *"has_tv, has_internet, children_allowed"*. [4] They only exist in the DSTC 3 dataset and have few appearances in the seed data. A test set is chosen for specific evaluation on these new semantic slots, by gathering all the sentences (688 sentences) whose annotation contains these three slots and randomly selecting 1000 sentences irrelevant to these three slots from the *dstc3_test* set. This test set is named as **DSTC3_S_test** (1688 sentences).

The union of a slot and action is taken as a plain semantic slot (e.g. *"confirm.food=Chinese"*), since each slot is tied with an action (e.g. *"inform"*, *"deny"* and *"confirm"*) in DSTC 2&3. The slot and action are taken as atomic concepts. For the slot filling task, only the semantic annotation with aligned information is kept, e.g. the semantic tuple *"request(phone)"* is ignored. We use transcripts as input, and make slot-value alignment by

[4]For each slot of *"has_tv, has_internet, children_allowed"*, the semantic annotation *"request(slot)"* is replaced with *"confirm(slot=True)"*. Then we have the slot-tagging format, e.g. "does it have [television:confirm.has_tv]".

Reference	... could get in [boston:city_name] [late:**period_of_day**] [night:period_of_day]
PS	... could get in [boston:city_name] [late:**airport_name**] [night:period_of_day]
AC	... could get in [boston:city_name] [late:**period_of_day**] [night:period_of_day]
ACD	... could get in [boston:city_name] [late:**period_of_day**] [night:period_of_day]

Table 3: Examples show how concept transfer learning benefits. We use *[value:slot]* for annotation.

string matching simply.

6.2.1 Experimental Results and Analysis

The experimental settings are similar to the ATIS's, whereas the seed data in DSTC 3 is also used for validation.

Model	Training set	F_1-score
PS	dstc3_seed	83.52
PS	dstc2_train + dstc3_seed	89.57
AC	dstc3_seed	83.58
AC	dstc2_train + dstc3_seed	91.98
ACD	dstc2_train + dstc3_seed	**92.15**

Table 4: The performance of our methods evaluated on the DSTC3_S_test.

The performance of our methods in the DSTC 2&3 task is illustrated in Table 4. We can see that: 1) By incorporating the data of the source domain (dstc2_train), **PS** and **AC** achieve improvements respectively. 2) **AC** gains more than **PS** by modeling the plain semantic slot as atomic concepts. The atomic concepts promote the associated slots to share input features for the same atoms. 3) The atomic-concepts based slot filling considering the concepts dependently (**ACD**) gains little (0.17%) over **AC** considering the concepts independently. It may be due to the small size of dstc3_seed.

Case study: Several cases from these models (trained on the union set of dstc2_train and dstc3_seed) are also chosen to explain why the atomic-concepts based slot filling outperforms the typical plain slot filling, as shown in Table 5. From the above part of Table 5, we can see **PS** predicts a wrong slot. Because the grammar *"does it have [something]"* is only for the plain slot *"confirm.hastv"* in the seed data. From the below part of Table 5, we can see that only **ACD** which considers the concepts dependently predicts the right slot. Since *"confirm.childrenallowed"* never exists in the seed data, **PS** can't learn patterns about it. Limited by the quantity of the seed data, **AC** also doesn't extract the semantics correctly.

Reference	does it have [internet:confirm.hasinternet]
PS	does it have [internet:confirm.hastv]
AC	does it have [internet:confirm.hasinternet]
ACD	does it have [internet:confirm.hasinternet]

Reference	do they allow [children:confirm.CA]
PS	do they allow [children:CA]
AC	do they allow [children:CA]
ACD	do they allow [children:confirm.CA]

Table 5: Examples show how concept transfer learning benefits. CA denotes *childrenallowed*.

7 Conclusion

To address data sparsity problem of language understanding (LU) task, we present a novel method of concept definition based on well-defined atomic concepts. We present the concept transfer learning for slot filling on the atomic concept level to solve the problem of adaptive LU. The experiments on the ATIS and DSTC 2&3 datasets show our method obtains promising results and outperforms the traditional slot filling, due to the knowledge sharing of atomic concepts.

The atomic concepts are constructed manually in this paper. In future work, we want to explore more flexible concept definition for concept transfer learning of LU. Moreover, we also propose a competitive method based on slot name embedding which can be extracted from the literal description of the slot name automatically. The experimental result shows that it lays foundation for finding a more flexible concept definition method for adaptive LU.

Acknowledgments

This work has been supported by the China NSFC project (No. 61573241), Shanghai International Science and Technology Cooperation Fund (No. 16550720300) and the JiangSu NSFC project (BE2016078). Experiments have been carried out on the PI supercomputer at Shanghai Jiao Tong University. We also thank Tianfan Fu for comments that greatly improved the manuscript.

References

Emmanuel Ferreira, Bassam Jabaian, and Fabrice Lefvre. 2015. Zero-shot semantic parser for spoken language understanding. In *16th Annual Conference of the International Speech Communication Association (InterSpeech)*.

Alex Graves. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg.

L Heck and D Hakkani-Tur. 2012. Exploiting the semantic web for unsupervised spoken language understanding. In *Spoken Language Technology Workshop*, pages 228–233.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1995. The atis spoken language systems pilot corpus. In *Proceedings of the Darpa Speech and Natural Language Workshop*, pages 96–101.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2013. Dialog state tracking challenge 2 & 3. [online] Available: http://camdial.org/mh521/dstc/.

Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging sentence-level information with encoder lstm for semantic slot filling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2077–2083, Austin, Texas. Association for Computational Linguistics.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In *17th Annual Conference of the International Speech Communication Association (InterSpeech)*.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775.

Roberto Pieraccini, Evelyne Tzoukermann, Zakhar Gorelov, J-L Gauvain, Esther Levin, C-H Lee, and Jay G Wilpon. 1992. A speech understanding system based on statistical representation of semantics. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 193–196. IEEE.

Gokhan Tur, Dilek Hakkani-Tür, and Larry Heck. 2010. What is left to be understood in atis? In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 19–24. IEEE.

Ngoc Thang Vu. 2016. Sequential convolutional neural networks for slot filling in spoken language understanding. In *17th Annual Conference of the International Speech Communication Association (InterSpeech)*.

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.

Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. 2013. Recurrent neural networks for language understanding. In *INTERSPEECH*, pages 2524–2528.

Majid Yazdani and James Henderson. 2015. A model of zero-shot learning of spoken language understanding. In *Conference on Empirical Methods in Natural Language Processing*, pages 244–249.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 678–687.

Feifei Zhai, Saloni Potdar, Bing Xiang, and Bowen Zhou. 2017. Neural models for sequence chunking. In *AAAI*, pages 3365–3371.

Xiaodong Zhang and Houfeng Wang. 2016. A joint model of intent determination and slot filling for spoken language understanding. In *the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*.

Su Zhu, Lu Chen, Kai Sun, Da Zheng, and Kai Yu. 2014. Semantic parser enhancement for dialogue domain extension with little data. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 336–341. IEEE.

Su Zhu and Kai Yu. 2017. Encoder-decoder with focus-mechanism for sequence labelling based spoken language understanding. In *IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, pages 5675–5679.