Lexicon-injected Semantic Parsing for Task-Oriented Dialog

Xiaojun Meng\textsuperscript{1}, Wenlin Dai\textsuperscript{1,2,†}, Yasheng Wang\textsuperscript{1}, Baojun Wang\textsuperscript{1}, Zhiyong Wu\textsuperscript{2}, Xin Jiang\textsuperscript{1}, Qun Liu\textsuperscript{1}

\textsuperscript{1} Noah’s Ark Lab, Huawei Technologies
\textsuperscript{2} Shenzhen International Graduate School, Tsinghua University, Shenzhen, China

\{xiaojun.meng, wangyasheng, puking.w, Jiang.Xin, qun.liu\}@huawei.com, \{dwl20\}@mails.tsinghua.edu.cn, \{zywu\}@sz.tsinghua.edu.cn

\section*{ABSTRACT}

Recently, semantic parsing using hierarchical representations for dialog systems has captured substantial attention. Task-Oriented Parse (TOP), a tree representation with intents and slots as labels of nested tree nodes, has been proposed for parsing user utterances. Previous TOP parsing methods are limited on leveraging lexicon resources, which are often used to guide the real dialog system. To mitigate this issue, we first propose a novel span-splitting representation for span-based parser that outperforms existing methods. Then we present a novel lexicon-injected semantic parser, which collects slot labels of tree representation as a lexicon, and injects lexical features to the span representation of parser. An additional slot disambiguation technique is involved to remove inappropriate span match occurrences from the lexicon. Experiments show that our best parser produces a new state-of-the-art result (87.62\%) on the TOP dataset, and also confirm the effectiveness of our proposed lexicon-injected parser and slot disambiguation model.

\textbf{Index Terms---} semantic parsing, task-oriented parse, lexicon-injected, slot disambiguation

\section{1. INTRODUCTION}

Traditional direct classification and slot-filling methods \cite{1, 2} are widely used in dialog systems to parse task-oriented utterances, however, whose representation is flat and limited. It is usually composed of a single intent per utterance and at most one slot label per token. It’s difficult for such a flat representation to handle compositional and nested queries. For instance, an intent is included inside a slot, which is common in commercial conversational systems with multiple backend services \cite{3}.

To overcome the limitation of classical intent-slot frameworks, Gupta et.al proposed a hierarchical Task-Oriented Parsing (TOP) representation allowing slots to contain nested intents \cite{4}, as presented in Figure 1. They empirically show the TOP representation is expressive enough to model the vast majority of human-generated complex queries in given domains. Furthermore, this representation is easier to annotate and parse than alternatives such as logical forms.

A large amount of semantic parsers \cite{3, 5, 6, 7} are designed on this task-oriented representation. In particular, given an utterance $x = (x_0, x_1, \ldots, x_{n-1})$ with $n$ tokens, existing parsers are able to parse $x$ into the tree representation, as illustrated in Figure 1. In real dialog system, a couple of slot lexicon tables are often used to guide slot-filling. For instance, restaurants and locations nearby are known in advance and updated frequently in the lexicon table to guide phone assistants. However, existing work has not put efforts on leveraging the lexicon information to address task-oriented parsing.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\caption{Hierarchical representation of an utterance from TOP dataset (IN: = intent; SL: = slot).}
\end{figure}

Lexicon-based methods have demonstrated capabilities in various tasks such as sentiment analysis \cite{8} and NER \cite{9}. The lexicon-based method is up-and-coming for task-oriented semantic parsing with the concept of slot, since slot values are easily and meaningfully collected as a slot lexicon. We are thus motivated to design a lexicon-injected parser that is capable of leveraging lexicon resources to improve parsing utterances in real dialog system.

Inspired by Pasupat et.al \cite{3}, we propose a span-based and lexicon-injected parser that embeds each span token as a vector to predict the labels of the tree nodes covering this span. This label predication uses independent scoring of span representation, which is further enriched via either the splitting feature from its child nodes (section 2.4), or the valuable lexical feature from matching utterances to a collected slot lexicon (section 2.5). In particular, we tag each matched span as the slot category it belongs to, and propose a slot disambiguation model to remove slot mismatch occurrences. Our contributions in this paper are two-folds:

1. We present a span-based semantic parser that sets state of the art in parsing TOP representation, and incorporates splitting feature to represent each parent span as a root of subtree.
2. We propose a novel lexicon-injected method to further improve task-oriented semantic parsing, via the use of slot disambiguation technique to remove slot mismatch occurrences in the lexicon. Empirical studies demonstrate the effectiveness of our methods and its potentials in real dialog system.

\section{2. METHODOLOGY}

Our base model is a span-based parser as firstly used in \cite{10} for constituency parsing. A constituency tree over an utterance is a col-
lecion of labeled spans. Given an utterance, the task is to predict a parse tree. Span \((i, j)\) corresponds to the constituent that is located between position \(i\) and \(j\) in an utterance. The label of a span is either an intent or a slot (prefixed with IN: or SL: in Figure 1). Our parser follows an encoder-decoder architecture (Figure 2). The encoder is a two-layer transformer [11] and the decoder is a chart parser borrowed from [10].

2.1. Encoder

We first represent each word \(x_i\) using two pieces of information: an external context-aware word representation \(w_i\) and a learned position embedding, where every position \(i \in 0, 1, \ldots, n - 1\) is associated with a vector \(p_i\). We concatenate these two embeddings to generate a representation of a word:

\[
x_i = [w_i; p_i]
\]

Span representation. To get the representation of span \((i, j)\) in the utterance, we first introduce a notion of boundary between each consecutive words \((x_{i-1}, x_i)\) in the utterance. We feed the entire utterance \(x = (x_1, \ldots, x_n)\) into a two-layer Transformer to calculate the boundary representation as \(f_{i-1,i}\). Specifically, given the output \(h_i\) from the encoder for word \(x_i\), we split it into two halves \(h_{i-1}^1; h_i^1\) and define \(f_{i-1,i} = [h_{i-1}^1; h_i^1]\), which means each boundary representation is decided by the left and right context together.

For an utterance \(x\) with length as \(n\), we are able to calculate the context-aware boundary representations \(f_{0,1}, f_{1,2}, \ldots, f_{n-2,n-1}\) with size of \(n - 1\). Notice that the beginning and ending tokens are special tokens such as [CLS] and [SEP] used in [12].

We define the representation of span \((i, j)\) as:

\[
r_{i,j} = f_{j,j+1} - f_{i-1,i}
\]

which uses the left and right boundaries to decide the in-between span representation.

2.2. Decoder

We use the chart parser [10] with additional modifications [13, 14]. Our parser assigns a score function \(s(T)\) to each mapping \(T\), which can be regarded as a tree shown in Figure 1. It decomposes as:

\[
s(T) = \sum_{T((i,j))=l} s(i,j,l)
\]

where \(s(i,j,l)\) is a real-valued score for the span \((i, j)\) that has label \(l\). This span-label scoring function is implemented as a one-layer feedforward network, taking as input the span representation \(r_{i,j}\) and producing as output a vector of label scores:

\[
\begin{align*}
    s_{\text{label}}(i, j) &= \text{ReLU}(W r_{i,j} + b), \quad (1) \\
    s(i, j, l) &= [s_{\text{label}}(i, j)]_l \quad (2)
\end{align*}
\]

Let \(k\) denote the index of the splitting point where span \((i, j)\) is divided into left and right child nodes, we define \(s^*(i, j)\) as the score of the best subtree spanning \((i, j)\):

\[
s^*(i, j) = \max_{l,k} [s(i, j, l) + s^*(i, k-1) + s^*(k, j)] \quad (3)
\]

For the simplest case, the leaf node is a single word of an utterance. We calculate \(s^*(i, i+1)\) as:

\[
s^*(i, i+1) = \max_l [s(i, i+1, l)] \quad (4)
\]

To parse the full utterance \(x\), we compute \(s^*(1, n)\) and then traverse backward to recover the tree achieving that score. Therefore, the optimal inference tree \(T^*\) can be found efficiently using a CKY-style bottom-up inference algorithm:

\[
T^* = \arg \max_T s(T)
\]

2.3. Training Objective

To train the model, we use the margin loss as described in previous works [10]. Given the correct tree \(\hat{T}\) and predicted tree \(T\), the model is trained to satisfy the margin constraints:

\[
s(\hat{T}) \geq s(T) + \theta(\hat{T}, T) \quad (5)
\]

Here \(\theta\) is a distance function that measures the similarity of labeled spans between the prediction and the correct Tree. The margin loss is calculated as:

\[
\max[0, -s(\hat{T}) + \max_l [s(T) + \theta(\hat{T}, T)]] \quad (6)
\]

2.4. Span-splitting Representation

The majority of parsing models score each span independently, which is sub-optimal. To mitigate the independence assumption between node labels, Pasupat et al. [3] introduce edge scores: concatenates child node’s span embedding with its additional label embedding as input, and thus model the conditional distribution over all possible labels of its parent node. Via this method, edge scores connect a parent node label to its child node label.

However, a parent node is still unaware of the split point, where this parent span is divided into two child spans. Notice that in equation 3, the label scoring of parent span \(s(i, j, l)\) is unaware of the underlying split point \(k\). We propose a simple but effective way of
incorporating this splitting feature into the decision, without introducing a new embedding like edge scores. We define a new splitting representation \( \hat{r}_{i,j} \), that adds the boundary representation at the splitting point to the parent:

\[
\hat{r}_{i,j} = r_{i,j} + f_{k^* - 1, k^*}
\]

where \( k^* \) is the best splitting point dynamically computed as follows, dividing the parent span into two separate spans \((i, k - 1)\) and \((k, j)\):

\[
k^* = \arg \max_k [s^*(i, k - 1) + s^*(k, j)]
\]

With this new representation \( r \), each span becomes dynamic and more expressive as a root of the subtree it contains. Even for the same span, this new span representation with added splitting feature, can be different if its subtree structure varies. Notice that this modification has no effect on the decoder since we are using the dynamic programming algorithm to perform bottom-up CKY decoding.

### 2.5. Lexicon-injected Method

Motivated by existing works [8, 9] on leveraging lexicon resources to improve related task performances, we believe collecting a slot lexicon table beforehand and injecting the related lexical features to parsing models would also help.

#### 2.5.1. Lexicon Collection and Matching

We first collect a lexicon table: in the TOP train set, there are 36 distinct slot categories, and 20059 slot values (15238 are unique), which means each slot category has 557 ones on average. The lexicon is only built from train set since we have no clue of test set. When a span of utterance is included in some slot-category of the lexicon, we tag each word in the span as this slot-category. Therefor, as in Figure 3, one word may have multiple tags that are added together. For those words without any tag, we use an out-of-category icon is only built from train set since we have no clue of test set.

We first collect a lexicon table: in the TOP train set, there are 36 distinct slot categories, and 20059 slot values (15238 are unique), which means each slot category has 557 ones on average. The lexicon table beforehand and injecting the related lexical features to the input embedding for lexicon-injected parsers.

#### 2.5.2. Generalized Representation

In addition, to empower the slot-category embedding to affect parsing tasks, we propose to replace \( w_i \) with \( q_i \) when this word appears in the lexicon as shown in equation (8). In this generalized representation, we abandon the original word embeddings and enforce the slot-category embedding to take control of representing spans from the lexicon. Via this replacement, we also aim to make the model more informative of slot categories and less constrained to specific slot values.

\[
x_i = \begin{cases} 
[w_i; p_i; q_i], & t_i = t_o \\
[q_i; p_i; q_i], & \text{otherwise}
\end{cases}
\]

However, slot categories are often overlapping on words and spans. As in Table 1, given the context of utterances, a word or span may mismatch slot entries in the lexicon. This mismatch brings unexpected noises to the \( q_i \), especially in the case of nested hierarchical representation like the TOP dataset. Therefore, we further propose a disambiguation technique that aims to remove inappropriate slot matched occurrences.

#### 2.5.3. Slot Disambiguation

We regard this lexicon-mismatch problem as a sequence binary classification with the input of utterance, slot category and slot position, and output whether this slot match occurrence is correct or not in the given context. As shown in Table 1, for each utterance, the matched slot entries associated with the right position as annotated in the given context. As shown in Table 1, for each utterance, the matched slot entries associated with the right position as annotated in the parse tree is labeled as True (positive sample); otherwise, it’s a mismatch labeled as False (negative sample).

In the disambiguation model, we insert the slot category as a token to the left and right boundary of corresponding slot values in the origin utterance (see colorful tokens in Figure 3: each slot category composes two unused slot-category tokens - the left and right to the slot value) and feed them as a whole into the pretrained context-aware BERT [12] to perform sequence classification. The context-aware word embedding and slot category embedding keep updated in this training session. Finally, we use the [CLS] hidden state \( h_{cls} \) to perform sequence classification on label \( c \) as commonly used [12]:

\[
p(c|h_{cls}) = \text{softmax}(W h_{cls} + b)
\]

In the inference, we compare each utterance to the collected lexicon and find all the match occurrences. Then we use this disambiguation model to classify each match occurrence and remove inappropriate ones given the context. This filter aims to largely reduce...
We evaluate our proposed models on the TOP dataset [4], which has a hierarchical representation of intent-slot annotations for utterances in navigation and event domains.¹ We will release the codes soon.

### 3. EXPERIMENT

#### 3.1. Evaluation on TOP Representation

**Baselines.** Most baselines for this task are well described in [4, 6]. We only include the most competitive baselines in the Table 2: the first span-based parser on TOP representation [3], with an additional improvement of using edge scores to model relations between parent and child labels; the generative model Seq2SeqPtr [7] based on the Pointer-Generator architecture to understand user queries; a family of Seq2Seq models (decoupled RoBERTa/BART) [6] that set state of the art in parsing decoupled TOP representation.

We evaluate on a few variants of models: 1) base model with BERT-base [12] or RoBERTa-base [15] as contextualized word embedding; 2) adding splitting feature to span representation as $\hat{v}_{i,j}$; 3) the trivial lexicon-injected parser without using slot disambiguation; it means there exists an amount of noisy slot mismatch for $q_v$ embedding; 4) the lexicon-inject parser with oracle hints from the ground-truth parse tree; it means there is no slot mismatch; 5) the lexicon-injected parser with slot disambiguation technique; For 4) and 5), we use an additional generalized representation to empower the slot-category embedding. Once we use an external pretrained model with a different dimension of word embedding to $v_w$, we simply apply a learned one-layer feedforward network to align it.

#### 3.2. Model Performance

We report the exact match accuracy and the labeled bracket F1 score as widely measured for the parse tree constituents [16]. Our base model with the transformer encoder outperforms span-based baselines in two measurements (83.06/94.23%), as shown in Table 2.

| Method                      | Acc | F1  |
|-----------------------------|-----|-----|
| **Non-lexicon-injected parser:** |     |     |
| Pasupat                     | 80.80 | 93.35 |
| Pasupat-edge                | 81.80 | 93.63 |
| Decoupled RoBERTa           | 84.52 | -    |
| Decoupled BART              | 87.10 | -    |
| Seq2SeqPtr (+BERT)          | 83.13 | -    |
| Seq2SeqPtr (+RoBERTa)       | 86.67 | -    |
| Ours (base)†                | 83.06 | 94.23 |
| Ours (+Split)†              | 83.97 | 94.55 |
| Ours (+RoBERTa)             | 85.77 | 95.24 |
| **Our lexicon-injected parser:** |     |     |
| w/o Slot Disambiguation†    | 81.83 | 93.87 |
| w/ Slot Disambiguation†     | 85.63 | 96.13 |
| w/ SD + GR†                 | 86.80 | 96.34 |
| w/ SD + GR + RoBERTa        | **87.62** | **96.60** |

Table 2. Comparison of complete match accuracy and labeled bracket F1 of different methods on TOP test set. SD, GR and † denote slot disambiguation, generalized representation and use BERT-base model.

### 3.2.1. Lexicon-injected parser

The trivial lexicon-injected parser without using slot disambiguation technique (w/o SD) is not even comparable (81.83%) to the base model, because utterances highly overlap with the lexicon, which brings an amount of unexpected mismatch to the input embedding.

Results in Table 2 show the slot disambiguation technique (w/ SD) is very promising to remove inappropriate span match occurrences, and thus largely improves the downstream parsing accuracy from 83.06% to 85.63%. Moreover, the additional generalized representation (SD + GR) brings an improvement (+1.17 from 85.63% to 86.80%) to our parser. We believe GR empowers the weight of lexical feature on parsing decisions, and thus makes our parser be more informative of slot categories when parsing slot values. Overall, our best parser (SD + GR + RoBERTa) achieves the new state of the art (87.62%), which is even better than decoupled BART and an ensemble of RNNs [5].

#### 3.2.2. Slot disambiguation

Table 3 describes the accuracy of our slot disambiguation technique. Overall, our slot disambiguation performs fairly well on different test sets. It demonstrates a promising ability to remove inappropriate slot match entries in the given context of utterances, thus reducing noises in subsequent parsing models. This slot disambiguation technique may also be promising for a wide use in other NLP tasks such as NER [17]. In addition, even if the disambiguation model fails to remove negative match entries, the later parser still holds the possibility to save it and produces a correct parsing result.

| TOP Dataset  | Accuracy |
|--------------|----------|
| Dev-matched set | 98.42    |
| Test-matched set | 98.26    |

Table 3. Accuracy of binary classification in slot disambiguation. Each set includes all matched span entries of comparing utterances to the lexicon table constructed according to the train set.

This might not be a surprising result given baselines [3] are mostly using biLSTMs as sequence embedder. We believe pretrained transformer embedders such as BERT are more semantic expressive than RNN models. We find that adding split-up information (+Split) to the embedding (span-splitting representation) is able to improve the complete match by almost one percent (+0.91%), which is quite significant for intent classification and slot-filling tasks. The result suggests bottom-up splitting decisions from child spans contribute to improve the parser.

In addition, our base model (+RoBERTa) outperforms decoupled RoBERTa by 1.25%, and achieves comparable performances to the modern generative Seq2SeqPtr [7]. The performance of the decoupled BART [6] is better while we believe the credit goes to its high-capacity pretrained encoder. We aim to compare those methods in the same encoder setting.

### 4. CONCLUSION

In conclusion, we provide a novel solution for dialog system (e.g., phone assistant), where a couple of slot lexicon tables are often used to guide on identifying slot entries. Our novel lexicon-injected method to semantic parsing takes one more step towards real-time task-oriented parsing, and our best parser sets a new state of the art on the TOP dataset. Our experiments show the usefulness of adopting lexicon-injected techniques to semantic parsing.

¹ http://fb.me/semanticparsingdialog
5. REFERENCES

[1] Gokhan Tur and Renato De Mori, *Spoken language understanding: Systems for extracting semantic information from speech*, John Wiley & Sons, 2011.

[2] Samuel Louvan and Bernardo Magnini, “Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey,” *In Proceedings of the 28th International Conference on Computational Linguistics*, 2020.

[3] Panupong Pasupat, Sonal Gupta, Karishma Mandyam, Rushin Shah, Mike Lewis, and Luke Zettlemoyer, “Span-based hierarchical semantic parsing for task-oriented dialog,” *In Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[4] Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis, “Semantic parsing for task oriented dialog using hierarchical representations,” *In Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[5] Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer, “Improving semantic parsing for task oriented dialog;” *In Conversational AI Workshop at NeurIPS*, 2019.

[6] Armen Aghajanyan, Jean Maillard, Akshat Shrivastava, Keith Diedrick, Michael Haeger, Haoran Li, Yashar Mehdad, Veselin Stoyanov, Anuj Kumar, Mike Lewis, and Sonal Gupta, “Conversational semantic parsing,” *In Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[7] Subendri Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza, “Don’t parse, generate! a sequence to sequence architecture for task-oriented semantic parsing,” *in Proceedings of The Web Conference 2020*, 2020, pp. 2962–2968.

[8] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede, “Lexicon-based methods for sentiment analysis,” *Computational linguistics*, 2011.

[9] Xiaonan Li, Hang Yan, Xipeng Qiu, and Xuanjing Huang, “Flat: Chinese ner using flat-lattice transformer,” *In Association for Computational Linguistics (ACL)*, 2020.

[10] Mitchell Stern, Jacob Andreas, and Dan Klein, “A minimal span-based neural constituency parser,” *In Association for Computational Linguistics (ACL)*, 2017.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *In North American Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

[13] David Gaddy, Mitchell Stern, and Dan Klein, “What’s going on in neural constituency parsers? an analysis,” *In North American Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

[14] Nikita Kitaev and Dan Klein, “Constituency parsing with a self-attentive encoder,” *In Association for Computational Linguistics (ACL)*, 2018.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.

[16] Ezra Black, Steven Abney, Dan Flickinger, Claudia Gdaniec, Ralph Grishman, Phil Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith L Klavans, et al., “A procedure for quantitatively comparing the syntactic coverage of english grammars,” *in Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*, 1991.

[17] Guillaume Lampe, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer, “Neural architectures for named entity recognition,” *in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.