Span-based Hierarchical Semantic Parsing for Task-Oriented Dialog

Panupong Pasupat[*]
Stanford University
ppasupat@cs.stanford.edu

Sonal Gupta
Facebook Assistant
sonalgupta@fb.com

Karishma Mandyam[*]
University of Washington
krm28@cs.washington.edu

Rushin Shah[*]
Google
rushinshah@google.com

Mike Lewis
Facebook AI Research
mikelewis@fb.com

Luke Zettlemoyer
Facebook AI Research
lsz@fb.com

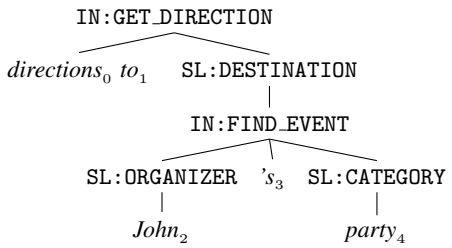
Abstract

We propose a semantic parser for parsing compositional utterances into Task Oriented Parse (TOP), a tree representation that has intents and slots as labels of nesting tree nodes. Our parser is span-based: it scores labels of the tree nodes covering each token span independently, but then decodes a valid tree globally. In contrast to previous sequence decoding approaches and other span-based parsers, we (1) improve the training speed by removing the need to run the decoder at training time; and (2) introduce edge scores, which model relations between parent and child labels, to mitigate the independence assumption between node labels and improve accuracy. Our best parser outperforms previous methods on the TOP dataset of mixed-domain task-oriented utterances in both accuracy and training speed.

1 Introduction

Most commercial conversational AI systems parse task-oriented utterances using intent classification and slot-filling models (He and Young, 2003; Raymond and Riccardi, 2007; Mesnil et al., 2015), where the intent is the task of the utterance (e.g., `IN:GET_DIRECTION`) and the slots are the parameters needed to complete the task (e.g., `SL:DESTINATION`). This limited representation typically allows only a single intent per utterance and at most one slot label per token. Dialog systems using such a flat representation would struggle to handle compositional tasks that involve invoking multiple backend services (e.g., "direction to John's party": find John's address, and then find the direction to that address).

To support compositional utterances, the hierarchical Task Oriented Parsing (TOP) representation has recently been introduced (Gupta et al., 2018). As illustrated in Figure 1, the TOP representation



$$T = \{(0,5) : (\texttt{IN:GET\_DIRECTION}),$$
$$(2,5) : (\texttt{SL:DESTINATION}, \texttt{IN:FIND\_EVENT}),$$
$$(2,3) : (\texttt{SL:ORGANIZER}),$$
$$(4,5) : (\texttt{SL:CATEGORY}),$$
$$(0,1) : \varnothing, \quad (0,2) : \varnothing, \quad (0,3) : \varnothing, \quad \dots\}$$

Figure 1: An example TOP tree and its mapping representation T. (`IN:` = intent; `SL:` = slot)

is a tree where intents and slots are nested alternatively to model composition. The values inside intent and slot subtrees can then be used by downstream dialog modules to invoke appropriate services in a hierarchical fashion.

We propose a span-based semantic parser for parsing utterances into the TOP representation. In its most basic form, the parser embeds each token span (e.g., $x_{2:5}$ = *"John 's party"* in Figure 1) as a vector, and then uses it to predict the labels of the tree nodes covering the span (e.g., `SL:DESTINATION` and `IN:FIND_EVENT`). While the label prediction is done independently for each span, a CKY decoding algorithm is used to decode a valid tree with the maximum tree score.

Our main contributions are twofold. First, we reinterpret the ad-hoc tree score in previous span-based parsing work (Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018) as a joint distribution over the labels. Under this new framework, the loss function factors nicely, which allows us to train the model in a highly parallelized fashion instead of having to run the computation-

[*] work done while at Facebook Assistant.

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 1520–1526, Hong Kong, China, November 3–7, 2019.

ally expensive decoder during training. Second, we introduce *edge scores* to model the relationship between parent and child nodes in the tree. This mitigates the independence assumption and allows the decisions at child nodes to influence the parent node.

We evaluate our models on the TOP dataset (Gupta et al., 2018) of compositional utterances on events and navigation domains. Most utterances have nested intents, with some utterances containing intents from different domains. We demonstrate that our model is fast to train and outperforms previous models on the dataset.[1]

2 Related work

Neural tree generation. Previous work on syntactic and semantic parsing employs different strategies for generating trees. Approaches such as transition-based parsers (Dyer et al., 2016; Liu and Zhang, 2017) and top-down tree generation (Vinyals et al., 2015; Choe and Charniak, 2016; Dong and Lapata, 2016; Krishnamurthy et al., 2017; Yin and Neubig, 2018) frame tree generation as predicting a sequence of actions for generating the tree. The decoding processing generally consists of local decisions, and beam search is used to retain uncertainty. In contrast, global decoding methods (Durrett and Klein, 2015; Lee et al., 2016) can incorporate non-local features and decode a tree with the maximum global score.

Span-based models. Span-based models use the embeddings of token spans to perform prediction. By capturing the properties of the whole phrase instead of individual words, span embeddings can be suitable for tasks where phrases are the basic unit. Indeed, span-based models have recently shown great promise by achieving state-of-the-art results for segmentation and tagging (Kong et al., 2016), coreference resolution (Lee et al., 2017) and semantic role labeling (He et al., 2018).

For syntactic parsing, span embeddings have been used to score actions in shift-reduce parsing (Cross and Huang, 2016), or to score the existence and labels of tree nodes in bottom-up and top-down parsing (Stern et al., 2017; Kitaev and Klein, 2018). The analysis by Gaddy et al. (2018) shows that the span embeddings can learn to capture various information, such as label correlation and structural agreement, which was traditionally modeled by grammars or lexical features.

3 Setup

Given an utterance $x = (x_0, \ldots, x_{n-1})$ with n tokens, the task is to predict a TOP parse tree, as illustrated in Figure 1. Each leaf node corresponds to a token x_i, while each non-terminal node covers some *span* (i, j) with tokens $x_{i:j} = (x_i, x_{i+1}, \ldots, x_{j-1})$. The label l of each non-terminal node is either an intent (prefixed with `IN:`) or a slot (prefixed with `SL:`). Two type constraints for TOP trees include: (1) intents can only have slots as children and vice versa, and (2) the tree root covering span $(0, n)$ must be an intent.

For our span-based parser, it is helpful to view the tree as a *mapping* T from each span (i, j) to a *chain* $c = (l_1, \ldots, l_k)$ which lists the $k \geq 0$ labels in the unary chain covering the span (i, j). For instance, the span $(2, 5)$ in Figure 1 has $T[2, 5] =$ (`SL:DESTINATION`, `IN:FIND_EVENT`). If no non-terminal node covers the span, we denote the empty chain as $c = \varnothing$.

Following previous span-based parser work (Cross and Huang, 2016; Stern et al., 2017), we only consider unary chains that appear in the training data to be valid chains. (In our experiments, the training data has 57 distinct labels and 135 distinct chains. Only a single chain in the test data is not covered in training data.)

4 Model

Our parser is based on a joint distribution over the chain at each span. Let the probability of span (i, j) having chain c be a softmax over the possible chains:

$$p(T[i, j] = c) = \frac{\exp\left[f_{\mathrm{n}}(x_{i:j}, c)\right]}{\sum_{c'} \exp\left[f_{\mathrm{n}}(x_{i:j}, c')\right]}. \quad (1)$$

(The conditioning on x is omitted.) Here, the *node score* $f_{\mathrm{n}}(x_{i:j}, c)$ is a real-valued function with $f_{\mathrm{n}}(x_{i:j}, \varnothing)$ fixed as 0. To compute the node scores for a span $x_{i:j}$, we embed the span using an LSTM-based span embedder[2] from Lee et al. (2017), and then apply a feedforward network to produce a real-valued score for each chain $c \neq \varnothing$.

We now make a simplifying assumption that the values of $T[i, j]$ are independent. The log-

[1] The code is available at `https://github.com/ppasupat/factored-span-parsing`.

[2] Refer to Appendix A for modeling details.

likelihood of the entire mapping T becomes

$$\begin{aligned}\log p(T) &= \sum_{i<j} \log p(T[i,j]) \\ &= \sum_{i<j} \begin{bmatrix} f_{\mathrm{n}}(x_{i:j}, T[i,j]) \\ -\log \sum_{c'} \exp\left[f_{\mathrm{n}}(x_{i:j}, c')\right] \end{bmatrix}.\end{aligned} \tag{2}$$

The log-sum-exp term $\log \sum_{c'} \exp f_{\mathrm{n}}(x_{i:j}, c')$ does not depend on T. During inference, maximizing the log-likelihood $\log p(T)$ is thus equivalent to maximizing the *tree score*:

$$s(T) := \sum_{i<j} f_{\mathrm{n}}(x_{i:j}, T[i,j]). \tag{3}$$

At test time, we use CKY chart parsing to decode a valid TOP tree T with the maximum score $s(T)$.

Training. The tree score $s(T)$ turns out to be the same scoring function used in previous span-based constituency parsing models (Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018). To train the model, these previous works use margin loss:

$$\max\left\{0, -s(T^*) + \max_{T}\left[\Delta(T, T^*) + s(T)\right]\right\} \tag{4}$$

where T^* is the gold tree, T is the predicted tree and Δ is a distance function. Computing the $\max_T$ term requires running a cost-augmented CKY decoder, which is computationally expensive and difficult to parallelize, especially when we add edge scores as described in the next section.

Instead, we propose to directly maximize the log-likelihood in Equation 2 of the gold trees in the training data. Concretely, given a gold tree T^*, our loss function is the negative log-likelihood $-\log p(T^*)$, which decomposes into a cross-entropy loss for each span (i, j) according to Equation 2. We train our model by directly minimizing these cross-entropy loss terms; in other words, we treat the model as a multiclass classification model where the input is the span $x_{i:j}$ and the classes are the possible chains for the span.

Our training method is faster than using margin loss since it does not require running the CPU-bound CKY decoder during training. Moreover, the cross-entropy loss terms can be computed in parallel for all spans of *multiple examples* at once, which leads to further speed-up.

Class weight for empty chains. In practice, the number of spans (i, j) with gold chains $T^*[i,j] = \varnothing$ (i.e., no subtree covering the span) is large compared to the total number of spans. To avoid the class imbalance problem, we scale the cross-entropy loss terms for spans with $T^*[i,j] = \varnothing$ by a hyperparameter $\alpha < 1$.

5 Edge scores

The model so far scores each span independently, which can be sub-optimal. For example, the prediction of the top intent only depends on the embedding of $x_{0:n}$, which is equivalent to standard intent classification or the first decision of a top-down tree generation. Similarly, ontology constraints (which intents can take which slots) are also not captured by the model.

To allows child nodes to influence the decision of the parent at inference time, we introduce *edge scores*. For a span (i, j) with $T[i,j] = c = (l_1, \ldots, l_k)$, consider the parent node of the top-most non-terminal l_1, and define $\pi[i,j]$ to be its label. We model the conditional distribution of $\pi[i,j]$ as a softmax over all possible labels:

$$\begin{aligned}&p(\pi[i,j] = l \mid T[i,j] = c) \\ &\quad = \frac{\exp\left[f_{\mathrm{e}}(x_{i:j}, c, l)\right]}{\sum_{l'} \exp\left[f_{\mathrm{e}}(x_{i:j}, c, l')\right]}\end{aligned} \tag{5}$$

where $f_{\mathrm{e}}(x_{i:j}, c, l)$ is the *edge score*. Similar to node scores, we compute $f_{\mathrm{e}}(x_{i:j}, c, l)$ by applying a feed-forward network on the concatenation of the embeddings of $x_{i:j}$ and c. For convenience of notation, we let $\pi[i,j] = \varnothing$ when the parent does not exist (i.e., when $T[i,j] = \varnothing$ or $(i,j) = (0,n)$). In those situations, we let $f_{\mathrm{e}}(x_{i:j}, c, l)$ be 0 for $l = \varnothing$ and $-\infty$ otherwise.

We define the joint log-likelihood of T and π:

$$\begin{aligned}&\log p(T, \pi) \\ &= \sum_{i<j} \Big[\log p(T[i,j]) + \log p(\pi[i,j] \mid T[i,j])\Big] \\ &= \sum_{i<j} \begin{bmatrix} f_{\mathrm{n}}(x_{i:j}, T[i,j]) \\ -\log \sum_{c'} \exp\left[f_{\mathrm{n}}(x_{i:j}, c')\right] \\ +f_{\mathrm{e}}(x_{i:j}, T[i,j], \pi[i,j]) \\ -\log \sum_{l'} \exp\left[f_{\mathrm{e}}(x_{i:j}, T[i,j], l')\right] \end{bmatrix}.\end{aligned} \tag{6}$$

Unlike in the original model, the last log-sum-exp term in Equation 6 *does* depend on the tree T, and thus has to be included in the tree score. Our

new tree score is

$$s(T) := \sum_{i<j} \left[\begin{array}{l} f_{\mathrm{n}}(x_{i:j}, T[i,j]) \\ + \log p(\pi[i,j] \mid T[i,j]) \end{array} \right] \quad (7)$$

where $p(\pi[i,j] \mid T[i,j])$ is the softmax over edge scores as defined in Equation 5. Note that we cannot replace node scores $f_{\mathrm{n}}(x_{i:j}, T[i,j])$ with their log-softmax this way since we want dummy nodes to contribute a score of 0 to the tree score.

To train the model, we again directly minimize the negative log-likelihood of the gold tree T^*. The loss function factors into a cross-entropy loss term for each span (i,j) and for each edge in T^*.

Discussion. The joint modeling of $T[i,j]$ and $\pi[i,j]$ is closely related to the parent annotation technique in syntactic parsing (Johnson, 1998; Klein and Manning, 2003; Petrov et al., 2006), where certain non-terminal labels are split into multiple labels based on their parents in the grammar rule (e.g., `VP` with parent `S` becomes `VP^S`). In our framework, we can view the label and edge scores (Equation 7) as a score over parent-annotated labels $(T[i,j], \pi[i,j])$:

$$s(T) = \sum_{i<j} f_{\mathrm{a}}(x_{i:j}, T[i,j], \pi[i,j]) \quad (8)$$

where $f_{\mathrm{a}}(x_{i:j}, \varnothing, \varnothing) = 0$. From our preliminary experiments, we found that modeling f_{a} as a combination of label and edge scores, as in Equation 7, is empirically better than directly applying a softmax over possible pairs $(T[i,j], \pi[i,j])$.

One analysis in Gaddy et al. (2018) shows that the span embedding of $x_{i:j}$ is powerful enough to identify the parent label $\pi[i,j]$ with high accuracy. However, in TOP semantic trees, parent and child nodes can have span boundaries that are far apart (e.g., the top intent covering span $(0, 15)$ might have a child slot covering span $(7, 9)$). As an alternative to increasing the power of the span embedder to handle long-range relations, which could lead to overfitting, adding edge scores is a simpler way to model relations between labels.

6 Experiments

We evaluate our models on the TOP dataset (Gupta et al., 2018) of hierarchical intent-slot annotations for utterances in navigation and event domains.[3] We use the dataset version with the noisy `IN:UNSUPPORTED_*` intents excluded, which contains 28410 training, 4032 development, and 8241 test examples. The main evaluation metric is exact match accuracy: the fraction of predicted parses that *exactly* match the annotated parses. We also report the labeled bracket F1 score for the parse tree constituents (Black et al., 1991).

Training details. We highlight crucial modeling decisions of our models and defer other details to Appendix B.

- For a fair comparison, we use the same token and sequence embedders as Gupta et al. (2018): 200-dimensional GloVe embeddings and 2-layer 164-dimensional biLSTMs. We expect contextual embeddings (Peters et al., 2018; Devlin et al., 2018) and a transformer-based sequence embedder (Kitaev and Klein, 2018) to further improve the results, as observed in Einolghozati et al. (2018).

- The class weight α for empty chains is tuned on the development data: $\alpha = 0.4$ for the basic model and $\alpha = 0.2$ for the model with edge scores. We will later demonstrate how different choices of α affect the results.

Baselines. We compare our method to existing approaches proposed for this task in Gupta et al. (2018): a shift-reduce parser based on Recurrent Neural Network Grammars (RNNG) (Dyer et al., 2016), and the best sequence-to-sequence model that produces linearization of the trees based on CNN utterance embeddings (Gehring et al., 2017).[4] We also consider the span-based parsers by Stern et al. (2017) with the additional improvements from Gaddy et al. (2018).[5] The parsers were designed for constituency parsing, are trained with cost-augmented margin loss, and use either a bottom-up CKY decoder (Stern-chart) or a top-down greedy decoder (Stern-greedy).

Accuracy. Table 1 shows the exact match accuracy and bracket scores on the test data. The span-based model without edge scores is comparable to the baseline RNNG model. With edge scores, the span-based model outperforms the baselines in both exact match accuracy and labeled F1 scores.

[3] http://fb.me/semanticparsingdialog

[4] https://github.com/facebookresearch/pytext

[5] https://github.com/mitchellstern/minimal-span-parser

Method	Acc	F1	Time
seq2seq	78.24	90.78	8m
RNNG	80.63	92.61	1h 16m
Stern-chart	80.66	93.03	25m
Stern-greedy	80.79	92.83	22m
ours (no f_e)	80.80	93.35	**5m**
ours (+ f_e)	**81.80**	**93.63**	9m

Table 1: The test exact match accuracy, labeled bracket F1, and training time per epoch of different methods.

Error type	no f_e	+ f_e
wrong top intent	106	99
wrong label except top intent	381	367
wrong non-terminal boundary	133	133
missing a non-terminal	153	154
spurious non-terminal	198	185
joining two gold non-terminals	20	26
splitting a gold non-terminal	27	36
other errors	6	4

Table 2: The error breakdown on the development data of our span-based models. Note that an example can have multiple errors.

Training speed. Our span-based models can be trained in a highly parallelized fashion without having to run the computationally expensive decoder. Table 1 shows the average wall-clock time used to train the model for one epoch over the training data.

Error analysis. We compare the errors made by the span-based parsers on the development set. Table 2 provides a breakdown of the error counts of the models on development data.

Recall trade-off. As described in Section 4, the hyperparameter α controls the weight of the loss terms for the class $c = \varnothing$ (i.e., not building a subtree for the span). As such, we can use α to trade off two types of errors: missing non-terminals (over-predicting $c = \varnothing$) and spurious non-terminals (under-predicting $c = \varnothing$). As shown in Figure 2, higher α encourages the model to predict $c = \varnothing$ more frequently, leading to more missing non-terminals (intents and slots) and fewer spurious ones. We can also tune α based on the downstream tasks. For instance, getting a higher slot recall using a small α is arguably better for dialog systems since other downstream modules (e.g., entity linker) can detect and discard spurious slots.

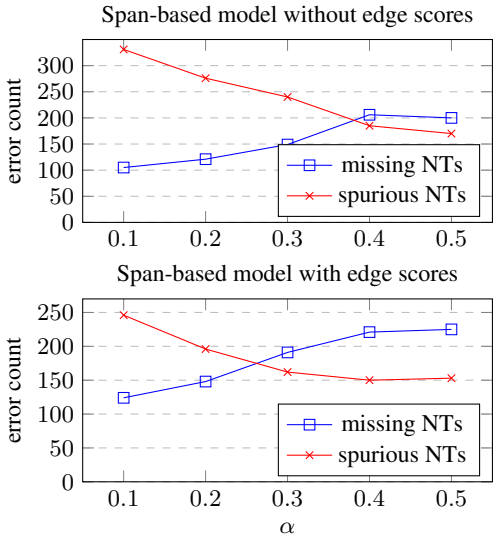


Figure 2: Trade-off between missing and spurious non-terminals with different α (weight for class $c = \varnothing$).

When α is low, one common type of errors is when the models incorrectly split a non-terminal. This usually happens when the gold slot consists of two sub-phrases that can be interpreted as slots on their own (e.g., a `SL:CATEGORY_EVENT` slot *"holiday concerts"* is split into a `SL:DATE_TIME` slot *"holiday"* and a `SL:CATEGORY_EVENT` slot *"concerts"*). As the tree score is the sum of span scores, the parser is biased toward creating two non-terminal nodes instead of one. Luckily, in the context of task-oriented dialogs, this type of errors tends to have small effects on the semantic interpretation, and sometimes even provides more useful information for downstream modules.

7 Conclusion

We presented the first span-based parser for parsing utterances into the hierarchical intent-slot representation. The log-likelihood objective allows us to train the model without having to decode a tree in a highly parallelized fashion, while edge scores can explicitly capture the parent-child relationship even when their boundaries are far apart.

Apart from standard accuracy improvement techniques such as better token embeddings and ensembling, possible future directions include a more fine-grained control of the recall trade-off, modeling the tokens outside the non-terminals instead of ignoring them, incorporating the parent's embedding in edge scores, and a more efficient or approximate decoder similar to the greedy decoder from Stern et al. (2017).

References

Ezra Black, Steven P. Abney, D. Flickenger, Claudia Gdaniec, Ralph Grishman, P. Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith L. Klavans, Mark Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Workshop on Speech and Natural Language*.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Empirical Methods in Natural Language Processing (EMNLP)*.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Association for Computational Linguistics (ACL)*.

Greg Durrett and Dan Klein. 2015. Neural crf parsing. In *Association for Computational Linguistics (ACL)*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *North American Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer. 2018. Improving semantic parsing for task oriented dialog. In *Conversational AI Workshop at NeurIPS*.

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

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning (ICML)*.

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

Luheng He, Kenton Lee, Omer Levy, and Luke S. Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling. In *Association for Computational Linguistics (ACL)*.

Yulan He and S. Young. 2003. A data-driven spoken language understanding system. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.

Mark Johnson. 1998. Pcfg models of linguistic tree representations. *Computational Linguistics*, 24.

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

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Association for Computational Linguistics (ACL)*.

Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Segmental recurrent neural networks. In *International Conference on Learning Representations (ICLR)*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Kenton Lee, Luheng He, Mike Lewis, and Luke S. Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Kenton Lee, Mike Lewis, and Luke S. Zettlemoyer. 2016. Global neural ccg parsing with optimality guarantees. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Jiangming Liu and Yue Zhang. 2017. Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association for Computational Linguistics (TACL)*, 5.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Z. Hakkani-Tür, Xiaodong He, Larry P. Heck, Gökhan Tür, Dong Yu, and Geoffrey Zweig. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Association for Computational Linguistics (ACL)*.

Christian Raymond and Giuseppe Riccardi. 2007. Generative and discriminative algorithms for spoken language understanding. In *INTERSPEECH*.

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

Oriol Vinyals, Lukasz Kaiser, Terry K Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems (NIPS)*.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Empirical Methods in Natural Language Processing (EMNLP)*.