Exploiting Rich Syntactic Information for Semantic Parsing with Graph-to-Sequence Model

Kun Xu[1]*, **Lingfei Wu**[2], **Zhiguo Wang**[2], **Mo Yu**[2], **Liwei Chen**[3], **Vadim Sheinin**[2]
[1]Tencent AI Lab
[2]IBM Research
[3]Peking University, Beijing, China
{syxu828,zgw.tomorrow,gflfof}@gmail.com, lwu@email.wm.edu
chenliwei@pku.edu.cn,vadims@us.ibm.com

Abstract

Existing neural semantic parsers mainly utilize a sequence encoder, i.e., a sequential LSTM, to extract word order features while neglecting other valuable syntactic information such as dependency or constituent trees. In this paper, we first propose to use the *syntactic graph* to represent three types of syntactic information, i.e., word order, dependency and constituency features; then employ a graph-to-sequence model to encode the syntactic graph and decode a logical form. Experimental results on benchmark datasets show that our model is comparable to the state-of-the-art on Jobs640, ATIS, and Geo880. Experimental results on adversarial examples demonstrate the robustness of the model is also improved by encoding more syntactic information.

1 Introduction

The task of *semantic parsing* is to translate text to its formal meaning representations, such as logical forms or structured queries. Recent neural semantic parsers approach this problem by learning soft alignments between natural language and logical forms from (text, logic) pairs (Jia and Liang, 2016; Dong and Lapata, 2016; Krishnamurthy et al., 2017). All these parsers follow the conventional encoder-decoder architecture that first encodes the text into a distributional representation and then decodes it to a logical form. These parsers may differ in the choice of the decoders, such as sequence or tree decoders, but they utilize the same encoder which is essentially a sequential Long Short-Term Memory network (SeqLSTM). This encoder only extracts word order features while neglecting useful syntactic information, such as dependency parse and constituency parse.

However, the syntactic features capture important structural information of the natural language input, which complements the simple word sequence. For example, a dependency graph presents grammatical relations that hold among the words; and a constituent tree provides a phrase structure representation. Intuitively, by incorporating such additional information, the encoder could produce a more meaningful and robust sentence representation. The combination of these features (i.e., sequence + trees) forms a general graph structure (see Figure 1). This inspires us to apply a graph encoder to produce a representation of a graph-structured input. The graph encoder also has the advantages that it could simultaneously encode all types of syntactic contexts, and incorporate multiple types of syntactic structures in a unified way.

In this paper, we first introduce a structure, namely *syntactic graph*, to represent three types of syntactic information, i.e., word order, dependency and constituency features (see §2). We then employ a novel graph-to-sequence (Graph2Seq) model (Xu et al., 2018), which consists of a graph encoder and a sequence decoder, to learn the representation of the syntactic graph (see §3). Specifically, the graph encoder learns the representation of each node by aggregating information from its K-hop neighbors. Given the learned node embeddings, the graph encoder uses a pooling-based method to generate the graph embedding. On the decoder side, a Recurrent Neural Network (RNN) decoder takes the graph embedding as its initial hidden state to generate the logical form while employing an attention mechanism over the node embeddings. Experimental results show that our model achieves the competitive performance on Jobs640, ATIS, and Geo880 datasets.

Different from existing works, we also investigate the robustness of our model by evaluating the model on two types of adversarial examples (Belinkov and Bisk, 2017; Cheng et al., 2018).

* Work done when the author was at IBM Research.

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 918–924
Brussels, Belgium, October 31 - November 4, 2018.

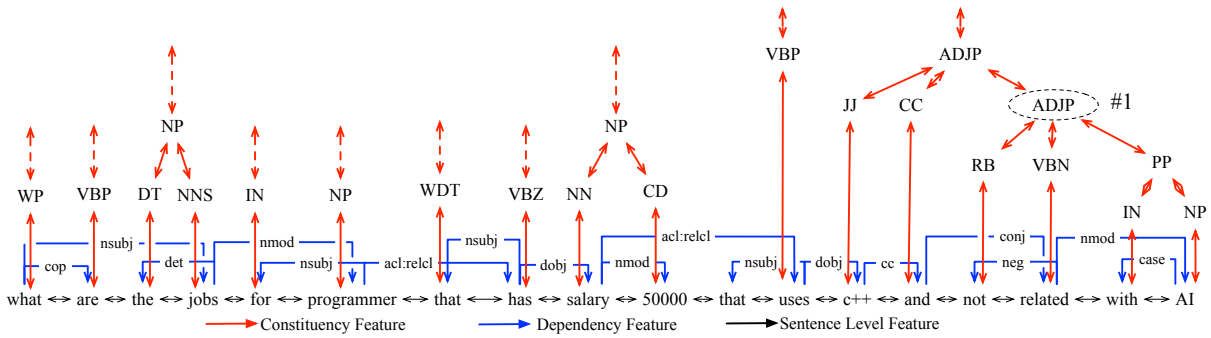

Figure 1: The syntactic graph for the Jobs640 question *what are the jobs for programmer that has salary 50000 that uses c++ and not related with AI*. Due to the space limitation, the constituent tree is partially shown here.

Experimental results show that the model coupling all syntactic features has the best robustness, achieving the best performance. Our code and data is available at https://github.com/IBM/Text-to-LogicForm.

2 Syntactic Graph

We represent three types of syntactic features, i.e., word order, dependency parse and constituency parse, as the syntactic graph (see Figure 1).

• **Word Order Features.** Previous neural semantic parsers mainly use these features by building a SeqLSTM that works on the word sequence. Our syntactic graph also incorporates this information by generating a node for each word and connecting them in the chain form. In order to capture the forward and backward contextual information, we link these nodes in two directions, that is, from left to right and from right to left.

• **Dependency Features.** A dependency parse describes the grammatical relations that hold among words. Reddy et al. (2016, 2017) have demonstrated that the dependency parse tree could be directly transformed to a logical form, which indicates that the dependency information (i.e., tree structure and dependency labels) is critical to the semantic parsing task. We incorporate this information into the syntactic graph by adding directed edges between the word nodes and assign them with dependency labels.

• **Constituency Features.** Similar to the dependency parse, the constituency parse represents the phrase structure, which is also important to the semantic parsing task. Take Figure 1 as an example: given the constituent tree that explicitly annotates "*not related with AI*" (node #1) is a proposition phrase, the model could learn a meaningful embedding for this phrase by encoding this structure into the model. Motivated by this observation, we add the non-terminal nodes of the constituent tree and the edges describing their parent-child relationships into the syntactic graph.

3 Graph-to-sequence Model for Semantic Parsing

After building the syntactic graph for the input text, we employ a novel graph-to-sequence model (Xu et al., 2018), which includes a graph encoder and a sequence decoder with attention mechanism, to map the syntactic graph to the logical form. Conceptually, the graph encoder generates node embeddings for each node by accumulating information from its K-hop neighbors, and then produces a graph embedding for the entire graph by abstracting all these node embeddings. Next, the sequence decoder takes the graph embedding as the initial hidden state, and calculates the attention over all node embeddings on the encoder side to generate logical forms. Note that this graph encoder does not explicitly encode the edge label information, therefore, for each labeled edge, we add a node whose text attribute is the edge's label.

Node Embedding. Given the syntactic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we take the embedding generation process for node $v \in \mathcal{V}$ as an example to explain the node embedding generation algorithm[1]:

(1) We first transform node v's text attribute to a feature vector, $\mathbf{a}_v$, by looking up the embedding matrix W_e;

(2) The neighbors of v are categorized into forward neighbors $\mathcal{N}_{\vdash}(v)$ and backward neighbors $\mathcal{N}_{\dashv}(v)$ according to the edge direction. In particular, $\mathcal{N}_{\vdash}(v)$ returns the nodes that v directs to and $\mathcal{N}_{\dashv}(v)$ returns the nodes that direct to v;

(3) We aggregate the **forward** representations of v's forward neighbors $\{\mathbf{h}_{u\vdash}^{k-1}, \forall u \in \mathcal{N}_{\vdash}(v)\}$ into

[1]Interested readers may refer to (Xu et al., 2018) for more implementation details.

a single vector, $\mathbf{h}^k_{\mathcal{N}_\vdash(v)}$, where $k \in \{1, ..., K\}$ is the iteration index. Specifically, this aggregator feeds each neighbor's vector to a fully-connected neural network and applies an element-wise max-pooling operation to capture different aspects of the neighbor set. Notice that, at iteration k, this aggregator only uses the representations generated at $k-1$. The initial forward representation of each node is its feature vector calculated in step (1);

(4) We concatenate v's current **forward** representation $\mathbf{h}^{k-1}_{v\vdash}$ with the newly generated neighborhood vector $\mathbf{h}^k_{\mathcal{N}_\vdash(v)}$. The resulted vector is fed into a fully connected layer with nonlinear activation function σ, which updates the **forward** representation of v, $\mathbf{h}^k_{v\vdash}$, to be used at the next iteration;

(5) We update the **backward** representation of v, $\mathbf{h}^k_{v\dashv}$ using the similar procedure as introduced in step (3) and (4) except that operating on the backward representations;

(6) We repeat steps (3)$\sim$(5) K times, and the concatenation of the final forward and backward representations is used as the final representation of v. Since the neighbor information from different hops may have different impacts on the node embedding, we learn a distinct aggregator at each iteration.

Graph Embedding. We feed the obtained node embeddings into a fully-connected neural network, and apply the element-wise *max*-pooling operation on all node embeddings. We did not find substantial performance improvement using *mean*-pooling.

Sequence Decoding. The decoder is an RNN which predicts the next token y_i given all the previous words $y_{<i} = y_1, ..., y_{i-1}$, the RNN hidden state s_i for time-step i and the context vector c_i that captures the attention of the encoder side. In particular, the context vector c_i depends on a set of node representations ($\mathbf{h}_1$,...,$\mathbf{h}_\mathcal{V}$) to which the encoder maps the input graph. The context vector c_i is dynamically computed using an attention mechanism over the node representations. The whole model is jointly trained to maximize the conditional log-probability of the correct description given a source graph. In the inference phase, we use the beam search algorithm to generate a description with beam size = 3.

4 Experiments

We evaluate our model on three datasets: Jobs640, a set of 640 queries to a database of job listings; Geo880, a set of 880 queries to a database of U.S. geography; and ATIS, a set of 5,410 queries to a flight booking system. We use the standard train/development/test split as previous works, and the logical form accuracy as our evaluation metric.

Method	Jobs	Geo	ATIS
Zettlemoyer and Collins (2007)	79.3	86.1	84.6
Kwiatkowski et al. (2011)	-	88.6	82.8
Liang et al. (2011)	90.7	87.9	-
Kwiatkowski et al. (2013)	-	**89.0**	-
Wang et al. (2014)	-	90.4	**91.3**
Zhao and Huang (2015)	85.0	88.9	84.2
Jia and Liang (2016)	-	85.0	76.3
Dong and Lapata (2016)-Seq2Seq	87.1	85.0	84.2
Dong and Lapata (2016)-Seq2Tree	90.0	87.1	84.6
Rabinovich et al. (2017)	**92.9**	85.7	85.3
Graph2Seq	91.2	88.1	85.9
w/o word order features	86.7	84.4	82.9
w/o dependency features	89.3	85.8	83.8
w/o constituency features	88.9	84.7	84.6
w/ ONLY word order features	88.0	84.8	83.1
BASELINE	88.1	84.9	83.0

Table 1: Exact-match accuracy on Jobs640, Geo880 and ATIS.

The model is trained using the Adam optimizer (Kingma and Ba, 2014), with mini-batch size 30. Our hyper-parameters are cross-validated on the training set for Jobs640 and Geo880, and tuned on the development set for ATIS. The learning rate is set to 0.001. The decoder has 1 layer, and its hidden state size is 300. The dropout strategy (Srivastava et al., 2014) with the ratio of 0.5 is applied at the decoder layer to avoid overfitting. W_e is initialized using GloVe word vectors from Pennington et al. (2014) and the dimension of word embedding is 300. For the graph encoder, the hop size K is set to 10, the non-linearity function σ is implemented as ReLU (Glorot et al., 2011), the parameters of the aggregators are randomly initialized. We use the Stanford CoreNLP tool (Manning et al., 2014) to generate the dependency and constituent trees.

Results and Discussion. Table 1 summarizes the results of our model and existing semantic parsers on three datasets. Our model achieves competitive performance on Jobs640, ATIS and Geo880. Our work is the first to use both multiple trees and the word sequence for semantic parsing, and it outperforms the Seq2Seq model reported in Dong and Lapata (2016), which only uses limited syntactic information.

Comparison with Baseline. To better demonstrate that our work is an effective way to utilize both multiple trees and the word sequence for semantic parsing, we compare with an addi-

tional straightforward baseline method (referred as BASELINE in Table 1). To deal with the graph input, the BASELINE decomposes the graph embedding to two steps and applies different types of encoders sequentially: (1) a SeqLSTM to extract word order features, which results in word embeddings, W_{seq}; (2) two TreeLSTMs (Tai et al., 2015) to extract the dependency tree and constituency features while taking W_{seq} as initial word embeddings. The resulted word embeddings and non-terminal node embeddings (from TreeLSTMs) are then fed into a sequence decoder.

We can see that our model significantly outperforms the BASELINE. One possible reason is that our graph encoder jointly extracts these features in a unified model by propagating the dependency and constituency information to all nodes in the syntactic graph. However, BASELINE separately models these features using two distinct TreeLSTMs. As a result, the non-terminal tree nodes only retain only one type of syntactic information propagated from their descendants in the tree.

Ablation Study. In Table 1, we also report the results of three ablation variants of our model, i.e., without word order features/dependency features/constituency features. We find that Graph2Seq is superior to Seq2Seq (Dong and Lapata, 2016) which is expected since Graph2Seq exploits more syntactic information. Among these features, the word order feature have more impact on the performance than other two syntactic features. By incorporating either the dependency or the constituency features, the model could gain further performance improvement, which underlines the importance of utilizing more aspects of syntactic information. Finally, removing both syntactic features (w/ ONLY word order) performs slightly worse compared to the Seq2Seq baseline. This shows that using K=10 hops is good enough for memorizing the sentences in our benchmarks, although still weaker compared to a bidirectional LSTM encoder.

A natural question here is on which type of queries our model could benefit from incorporating these parse features. By analyzing the queries and our predicted logical forms, we find that the parse features mainly improve the prediction accuracy for the queries with complex logical forms. Table 2 gives some running examples of complicated queries in three datasets. We find that the model that exploits three syntactic information could correctly predict these logical forms while the model that only uses word order features may fail.

Complicated Query & Predicted Logical Forms
Jobs Q: *what are the jobs for programmer that has salary 50000 that uses c++ and not related with AI*
Pred: answer(J,(job(J),-((area(J,R),const(R,'ai'))), language(J,L),const(L,'c++'), title(J,P), const(P,'Programmer'),salary_greater_than(J, 50000,year)))).
Geo Q: *which is the density of the state that the largest river in the united states run through*
Pred: answer(A,(density(B,A),state(B), longest(C,(river(C),loc(C,D),const(D,id(usa)))), traverse(C,B))))
ATIS Q: *please find a flight round trip from los angeles to tacoma washington with a stopover in san francisco not exceeding the price of 300 dollars for june tenth 1993*
Pred: (lambda $0 e (and (flight $0) (round_trip $0) (from $0 *los angeles*) (to $0 *tacoma washington*) (stop $0 *san francisco*) (< (cost $0) 300) (day_number $0 *tenth*) (month $0 *june*) (year $0 1993)))

Table 2: Examples of complicated query and predicted logical forms.

Robustness Study. Different from previous works, we evaluate the robustness of our model by creating adversarial examples with the hope to investigate the impact of introducing more syntactic information on robustness. Specifically, we create two types of adversarial examples and conduct experiments on the ATIS dataset. Following Belinkov and Bisk (2017), we first experiment with the synthetic noise, **SWAP**, which swaps two letters (e.g. noise→nosie). It is common to see such noisy information when typing quickly. Given a text, we randomly perform swap on $m \in \{1, 2, 3, 4, 5\}$ words that **not** correspond to the operators or arguments in logical forms, ensuring the meaning of the text is not changed. We train Graph2Seq on the training data and first evaluate it on the original development data, Dev_{ori}. Then we use the same model but evaluate it on a variant of Dev_{ori}, whose queries contain m swapped words.

Figure 2 summarizes the results of our model on the first type of adversarial examples, i.e., the ATIS development set with the SWAP noise. From Figure 2, we can see that (1) the performance of our model on all combinations of features degrade significantly when increasing the number of swapped words; (2) the model that uses three syntactic features (our default model) always achieves the best performance, and the performance gap

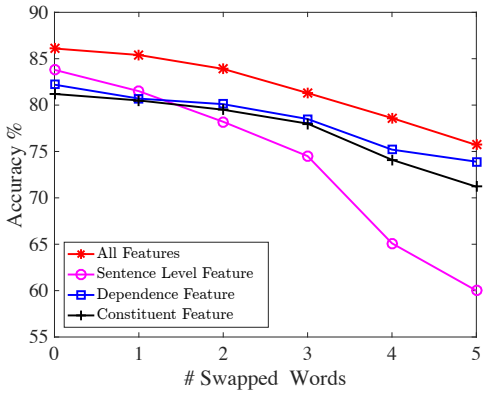

Figure 2: The logical form accuracy on the development set of ATIS with various swapped words number.

compared to others increases when rising the number of swapped words; (3) word order features are the most sensitive to the word sequence while the dependency and constituency features seem more robust to such noisy information; (4) thanks to the robustness of the dependency and constituency features, the default model performs significantly better than the one that only uses word order features on the noisy sentences. These findings demonstrate that incorporating more aspects of syntactic information could enhance the robustness of the model.

We also experiment with the paraphrase of the input text as the second type of adversarial examples. More specifically, we collect the paraphrase of a text by first translating it to the other language such as Chinese and then translating it back to English, using the Google Translate service. We use this method to collect a new variant of Dev_{ori} whose queries are the paraphrases of the original ones. By manually reading these queries, we find 94% queries convey the same meaning as original ones. Similar to the first experiment, we still train the model on Dev_{ori} and evaluate it on the newly created dataset.

Feature	Acc_{ori}	Acc_{para}	Diff.
Word Order	84.8	78.7	-6.1
Dep	83.5	80.1	-3.4
Cons	82.9	77.3	-5.6
Dep + Cons	84.0	80.7	-3.3
Word Order + Dep	85.2	82.3	-2.9
Word Order + Cons	84.9	79.9	-5.0
Word Order + Dep + Cons	**86.0**	**83.5**	**-2.5**

Table 3: Evaluation results on ATIS where Acc_{ori} and Acc_{para} denote the accuracy on the original and paraphrased development set of ATIS, respectively.

Table 3 shows the results of our model on the second type of adversarial examples, i.e., the paraphrased ATIS development set. We also report the result of our model on the original ATIS development set. We can see that (1) no matter which feature our model uses, the performance degrades at least 2.5% on the paraphrased dataset; (2) the model that only uses word order features achieves the worst robustness to the paraphrased queries while the dependency feature seems more robust than other two features. (3) simultaneously utilizing three syntactic features could greatly enhance the robustness of our model. These results again demonstrate that our model could benefit from incorporating more aspects of syntactic information.

5 Related Work

Existing works of generating text representation has evolved into two main streams. The first one is based on the word order, that is, either generating general purpose and domain independent embeddings of word sequences (Wu et al., 2018a; Arora et al., 2017), or building Bi-directional LSTMs over the text (Zhang et al., 2018). These methods neglect other syntactic information, which, however, has been proved to be useful in shallow semantic parsing, e.g., semantic role labeling (Punyakanok et al., 2008). To address this, recent works attempt to incorporate these syntactic information into the text representation. For example, Xu et al. (2016) builds separated neural networks for different types of syntactic annotation. Gormley et al. (2015); Wu et al. (2018b) decompose a graph to simpler sub-graphs and embed these subgraphs independently. Our approach, compared to the above methods, provided a unified solution to arbitrary combinations of syntactic graphs. In parallel to syntactic features, other works leverage additional information such as dialogue and paraphrasing for semantic parsing (Su and Yan, 2017; Gur et al., 2018).

6 Conclusions

Existing neural semantic parsers mainly leverage word order features while neglecting other valuable syntactic information. To address this, we propose to build a syntactic graph which represents three types of syntactic information, and further apply a novel graph-to-sequence model to map the syntactic graph to a logical form. Experimental results show that the robustness of our model is improved due to the incorporating more aspects of syntactic information, and our model outperforms previous semantic parsing systems.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*.

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. 2018. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *CoRR*, abs/1601.01280.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 315–323.

Matthew R Gormley, Mo Yu, and Mark Dredze. 2015. Improved relation extraction with feature-rich compositional embedding models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1774–1784.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1339–1349.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *CoRR*, abs/1606.03622.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*.

Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1512–1523.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. *Computational Linguistics*, 39:389–446.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*.

Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. *arXiv preprint arXiv:1702.03196*.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. *arXiv preprint arXiv:1704.05974*.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Adrienne Wang, Tom Kwiatkowski, and Luke S. Zettlemoyer. 2014. Morpho-syntactic lexical generalization for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1284–1295.

Lingfei Wu, Ian E.H. Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J. Witbrock. 2018a. Word mover's embedding: From word2vec to document embedding. In *EMNLP*.

Lingfei Wu, Ian En-Hsu Yen, Fangli Xu, Pradeep Ravikuma, and Michael Witbrock. 2018b. D2ke: From distance to kernel and embedding. *arXiv preprint arXiv:1802.04956*.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *ACL 2016*.

Kun Xu, Lingfei Wu, Zhiguo Wang, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL*.

Yue Zhang, Qi Liu, and Linfeng Song. 2018. Sentence-state lstm for text representation. *arXiv preprint arXiv:1805.02474*.

Kai Zhao and Liang Huang. 2015. Type-driven incremental semantic parsing with polymorphism. In *HLT-NAACL*.