An Empirical Study of Graph Neural Networks Based Semantic Parsing

Shucheng Li[*]
Nanjing University
shuchengli@smail.nju.edu.cn

Lingfei Wu[*]
IBM Research
wuli@us.ibm.com

Shiwei Feng
Nanjing University
swfeng98@gmail.com

Fengyuan Xu[†]
Nanjing University
fengyuan.xu@nju.edu.cn

Fangli Xu
Squirrel AI Learning
lili@yixue.us

Sheng Zhong
Nanjing University
sheng.zhong@gmail.com

ABSTRACT

Existing neural semantic parsers either only consider the word sequence for encoding or decoding, and ignore important syntactic information useful to parsing purposes. In this paper, we present a novel Graph Neural Networks (GNN) based neural semantic parser, namely Graph2Tree consisting of a graph encoder and a hierarchical tree decoder, that embraces the advantages of both worlds. In addition, we conduct a first study on the impacts of different syntactic graph constructions on the performance of GNNs for semantic parsing. We find that both noisy information and structural complexity introduced by a graph construction, due to imperfection of dependency tree parser or complex constituency tree, could lead to significantly detrimental effect on the performance of GNN-based semantic parsers.

CCS CONCEPTS

• **Computing methodologies** → Neural networks.

KEYWORDS

Graph Neural Networks, Semantic Parsing, Graph Construction

ACM Reference Format:
Shucheng Li, Lingfei Wu, Shiwei Feng, Fengyuan Xu, Fangli Xu, and Sheng Zhong. 2019. An Empirical Study of Graph Neural Networks Based Semantic Parsing. In *Proceedings of DLG '19: The First International Workshop on Deep Learning on Graphs: Methods and Applications (KDD Workshop DLG '19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

[*]Both authors contributed equally to this research.
[†]Corresponding author

KDD Workshop DLG '19, August 05, 2019, Anchorage, Alaska, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Semantic parsing is the task of translating natural language utterances into machine-interpretable meaning representations like logical forms or executable queries. Recent years have seen a surge of interests in developing neural semantic parsers with sequence-to-sequence models. These parsers have achieved tremendous contributions [7, 11, 15].

Due to the fact that the meaning representations are usually structured objects (e.g. tree structures), much efforts have been devoted to develop structure-oriented decoders, including tree decoders [1, 7], grammar constrained decoders [8, 12, 23, 24], action sequences for semantic graph generation [6], and modular decoders based on abstract syntax trees [18, 25]. Despite the impressive results these approaches have achieved, they only consider the word sequence information and ignore other rich syntactic information, such as dependency parsing tree or constituency tree, available at the encoder side.

Lately researchers have proved the important applications of the Graph Neural Networks (GNNs) in various NLP tasks, including the neural machine translation [3, 4], information extraction [26], and AMR-to-text [20]. In the semantic parsing field, a general Graph2Seq model [22] is proposed to incorporate these dependency and constituency trees with the word sequence and then create a syntactic graph as encoding input. However, their approach simply treats a logical form as a sequence, neglecting the rich information in a structured object like tree in the decoder architecture.

To address the aforementioned issues, we present a novel GNN-based neural semantic parser, namely Graph2Tree consisting of a graph encoder and a hierarchical tree decoder, that embraces the advantages of both worlds. Specifically, our graph encoder learns from a syntactic graph that could be constructed from both word sequence and the corresponding dependence parsing tree or constituency tree. Our tree

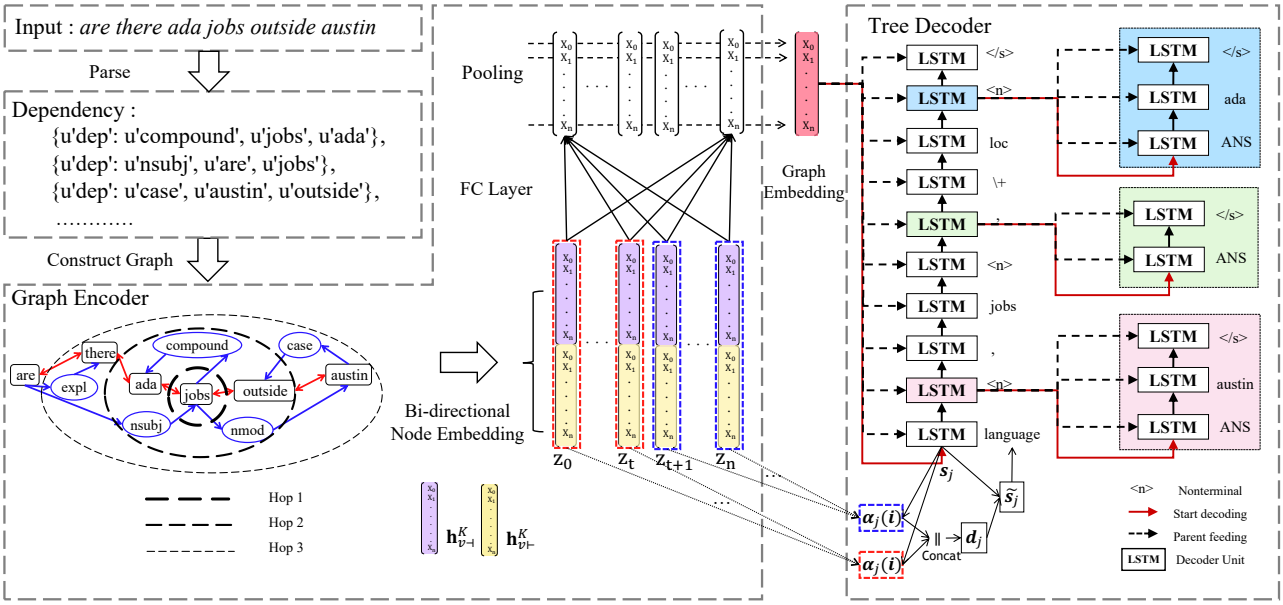

Figure 1: Overall Architecture of Graph2Tree Model

decoder decodes the logic forms from the learned graph-level vector representations to explicitly capture the compositional structure of logical form. We also introduce an attention mechanism to learn soft alignments between a syntactic graph and a logic form.

In addition, we conduct a first study on the impacts of different syntactic graph constructions on the performance of GNNs for semantic parsing. We find that both noisy information and structural complexity introduced by a graph construction, due to imperfection of dependency tree parser or complex constituency tree, could lead to significantly detrimental effect on the performance of GNN-based semantic parsers.

Our experiments on benchmark datasets *JOBS*, *GEO*, and *ATIS* demonstrate that, with right choices of constructed syntactic graph, our Graph2Tree model could match or outperform the performance of Seq2Seq, Seq2Tree, and Graph2Seq models.

2 GRAPH2TREE FOR SEMANTIC PARSING

We aim to learn a semantic parser that translates a syntactic graph and its logic form representation. As shown in Figure 1, our Graph2Tree model consists of a *graph encoder* that encodes a syntactic graph into a vector representation and a *tree decoder* that learns to generate logic form conditioned on the encoded graph-level vector.

Graph Encoder. In addition to the original sequence structure, their corresponding syntactic relations can be naturally incorporated into the input sequence which formulates an expressive graph data structure. As a result, we seek to exploit a graph neural network [14] to effectively learn high-quality vector representation from graph input. Specifically, we adapt the graph encoder in [21] that is an inductive node embedding algorithm. Conceptually, the model first generates the bi-directional node embedding (forward and backward embeddings). Each directional node embedding learns its vector representation by aggregating information from a node local neighborhood within K hops of the graph.

Given the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we first generate the feature vector $\mathbf{a}_v$ for all nodes $\forall v \in \mathcal{V}$ from v's text attribute using Long Short Term Memory (LSTM). These feature vectors are used as initial node embeddings. The graph encoder repeats the following process K times:

$$\mathbf{h}^0_{v\vdash} = \mathbf{a}_v, \mathbf{h}^0_{\vdash v} = \mathbf{a}_v, \forall v \in \mathcal{V} \tag{1}$$

$$\mathbf{h}^k_{\mathcal{N}_\vdash(v)} = \mathbf{M}^k_\vdash(\{\mathbf{h}^{k-1}_{u\vdash}, \forall u \in \mathcal{N}_\vdash(v)\}) \tag{2}$$

$$\mathbf{h}^k_{v\vdash} = \sigma(\mathbf{W}^k \cdot \texttt{CONCAT}(\mathbf{h}^{k-1}_{v\vdash}, \mathbf{h}^k_{\mathcal{N}_\vdash(v)})) \tag{3}$$

$$\mathbf{h}^k_{\mathcal{N}_\dashv(v)} = \mathbf{M}^k_\dashv(\{\mathbf{h}^{k-1}_{u\dashv}, \forall u \in \mathcal{N}_\dashv(v)\}) \tag{4}$$

$$\mathbf{h}^k_{v\dashv} = \sigma(\mathbf{W}^k \cdot \texttt{CONCAT}(\mathbf{h}^{k-1}_{v\dashv}, \mathbf{h}^k_{\mathcal{N}_\dashv(v)})) \tag{5}$$

where $k \in \{1, ..., K\}$ is the iteration index and $\mathcal{N}$ is the neighborhood function of node v. $\mathbf{M}^k_\vdash$ and $\mathbf{M}^k_\dashv$ are the forward and backward aggregator functions, $\mathbf{W}^k$ denotes weight matrices, σ is a non-linearity function. Node v's forward (backward) representation $\mathbf{h}^k_{v\vdash}$ ($\mathbf{h}^k_{v\dashv}$) aggregates the information of nodes in $\mathcal{N}_\vdash(v)$ ($\mathcal{N}_\dashv(v)$).

After the bi-directional node embeddings are learned, we can feed the obtained node embeddings into a fully-connected

neural network and apply the element-wise *max*-pooling operation on all node embeddings to compute the graph-level vector representation.

Tree Decoder As we mentioned in 1, the meaning representations we want to decode in semantic parsing task are usually structured (like tree structure). And actually, traditional sequence decoder take them as sequences and therefore need to memorize more redundant auxiliary information to recover the structure, which may bring some negative effects to accuracy.

To solve this problem, the goal of tree decoder is to faithfully learn the compositional nature of logic form representation. We choose to adapt the hierarchical tree decoder in [7] because of its simplicity and empirically good performance. As shown in Figure 1, the graph-level vector representation is encoded and then feed to the hierarchical tree decoder. Following [7], in our tree structure, we define "nonterminal" $\langle n \rangle$ token to start decoding a subtree and special tokens $\langle /s \rangle$ to denote the end of sequence. The tree decoder first uses LSTM to decode the logic form at depth 1 of the subtree that corresponds to parts of logical form t until token$\langle /s \rangle$ is predicted. And then it generates the parts of logical forms by conditioning on the corresponding nonterminal node's hidden representation. To fully utilize the nonterminal node's information, a parent-feeding scheme is employed to provide additional input of nonterminal node embedding to augment with the inputs and fed into LSTM.

Various attention mechanisms have been proposed [2, 16] to incorporate the hidden vectors of the inputs into account in the decoding processing. In particular, the context vector $\boldsymbol{s}_j$ depends on a set of node representations ($\mathbf{h}_1$,...,$\mathbf{h}_{\mathcal{V}}$) to which the encoder maps the input graph. In our model, different nodes(word nodes, compositional nodes et al.) construct the syntactic graph input. And it's a more natural way to give different attention on them. So we propose to compute separate attention mechanisms over different node representations corresponding to the original word tokens and the parsing tree nodes to compute the final context vector $\tilde{\boldsymbol{s}}_j$ for decoding tree structured outputs. Our model is then jointly trained to maximize the conditional log-probability of the correct description given a syntactic graph G.

3 CONSTRUCTING SYNTACTIC GRAPHS

To apply GNNs, the first step is to construct a syntactic graph. The syntactic graph is constructed from input word sequence with other syntactic information. How to construct such graphs is critical to this type of parser and influences the parsing results. In this section, we first introduce two existing methods for syntactic graph construction [22], and then discuss the potential issues of these constructed graphs due to imperfect dependency and constituent parse trees.

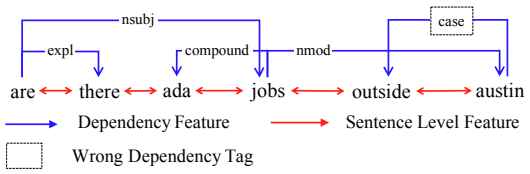

Figure 2: Graph with word sequence and dependency tree

Combining Word Sequence with Dependency Parse Tree. The dependency parse tree not only represents various grammatical relationships between pairs of text words, but also be shown that the dependency parse tree plays an important role in transforming texts into logical forms Reddy et al. [19]. Therefore, the first method integrates two types of features by adding dependency linkages between corresponding word pairs in word sequence. Concretely, we transform a dependency label into a node, and it is linked respectively with two word nodes which belong to it. Figure 2 illustrates an example of constructed syntactic graph from a text using this method.

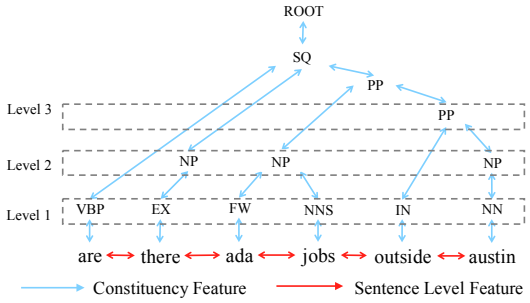

Figure 3: Graph with word sequence and constituency tree

Combining Word Sequence with Constituency Tree. The constituency tree contains the phrase structure information which is also critical to the describe the word relationships and has shown to provide useful information for translation [10]. Since the leaf nodes in the constituency tree are the word nodes in the text, this method merges these nodes with the identical ones in the bi-directional word sequence chain to create the syntactic graph. Figure 3 illustrates an example of constructed syntactic graph from a text using the second method.

Discussions on Graph Construction. Compared to the word sequence alone, these two types of graph construction clearly capture more syntactic structural information. However, as the graph structure becomes more complex, GNNs models are likely to be more sensitive to noises, compared to the word sequence.

The first observation is the noises in the dependency tree associated with the input texts. We find out that some dependency label could be incorrectly assigned due to the imperfection of the dependency tree parser. For example, the symbol '\+' in the logical form represents a negative meaning, so

words in the source text of dependency parse tree, such as *not, no, outside, exclude, and without*, should be linked it. Sometimes the word *outside* is nevertheless missed because the parser consider it does not have the negative meaning. Thus, we can see that the noise could be easily introduced into syntactic graph and usually this factor is ignored in previous work.

The other factor is the graph complexity. Sometimes the constituency tree generated is complicated with multiple layers. When the hops considered in the Graph2Tree and Graph2Seq parsers are not good enough, the proper word relationships probably will not be discovered and buried deeply due to the long path distance between two words. Instead, useless information may be considered in generating the logical form, leading to the wrong results. This factor is also not well studied in previous work so far.

4 EXPERIMENTS

We carry out experiments to evaluate the effectiveness of Graph2Tree, with the goal of answering: i) With what syntactic graphs, will GNN-based methods perform well? ii) With properly constructed graph inputs, does Graph2Tree perform better compared to baselines?

Datasets and preprocess. We evaluate our framework on three benchmark datasets JOBS, GEO, and ATIS. The first one is JOBS, a set of 640 qeuries to a database of job listings, the second is GEO, a set of 880 queries to a database of U.S. geography, while the last one ATIS is a set of 5410 quries to a flight booking system. We use the standard train/dev/test split as previoues works. For data preprocessing, We use the preprocessed version provided by Dong and Lapata [7], where natural language utterances are lowercased and stemmed with Bird et al. [5], and entity mentions are replaced by numbered markers. For graph construction, we use dependency parser and constituency parser from CoreNLP Manning et al. [17]. We choose the logical form accuracy as our evaluation.

Table 1: The impact of different graph constructions on performance of SP on *JOBS*. WO+DEP stands for Word Order + Dependency, and WO+CON stands for Word Order + Constituency, respectively

Methods	Graph2Tree	Graph2Seq
Word Order	91.4	87.1
Output correction: WO+DEP	92.1	88.6
Original condition: WO+DEP	91.5	85.0
Constituency two layer cut: WO+CON	91.4	86.4
Constituency one layer cut: WO+CON	**92.9**	88.6
Constituency tree all layer: WO+CON	92.1	85.0

Settings. We use Adam optimizer Kingma and Ba [13] and batch size is set to 20. In JOBS and GEO, our hyperparameters are cross-validated on the training set. For ATIS, we tuned them on the development set. The learning rate is set to 0.001. In graph encoder, the BiRNN we use is a one-layer bi-lstm with hidden size 150, and the hop size in GNN is choose from {2,3,4,5,6}. The decoder we employ is a one-layer lstm with hidden size 300. The dropout rate is chosen from {0.1,0.3,0.5}.We use Relu Glorot et al. [9] as our non-linear function and inference strategy in decoding logical forms is greedy search.

Table 2: The impact of different graph constructions on performance of SP on *GEO*

Methods	Graph2Tree	Graph2Seq
Word Order	87.1	84.6
Output Correction: WO+DEP	87.5	85.7
Original Condition: WO+DEP	87.1	84.3
Constituency two layer cut: WO+CON	87.5	84.3
Constituency one layer cut: WO+CON	**88.2**	85.7
Constituency tree all layer: WO+CON	87.1	83.9

Graph Constructions Schemes. For the noise factor, we test two conditions. The first one is the **original condition** which include the errors introduced by the dependency tree generator. For example, the CoreNLP sometimes does not correctly mark the word *outside* with the \+ symbol and therefore introduce noises to the graph. The second condition is called the **output correction**. In this condition, we manually remove all output results related to the negative meaning symbol, representing the best results we can obtain without any influence of noises introduced by the corresponding wrong tags of \+.

For the structural complexity, we also test three conditions w.r.t. *Word Order + Constituency* construction. We choose to remove different layers of non-terminal nodes in constituency tree to construct the syntactic graph with word sequence nodes. By constructing syntactic graph with different partial information in constituency tree, we can obtain syntactic graphs with various structural complexity.

Table 3: Exact-match accuracy comparison on *JOBS* and *GEO*

Methods	*JOBS*	*GEO*	*ATIS*
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**
Xu and Wu (2018)-Graph2Seq	88.6	85.7	83.8
Graph2Tree	**92.9**	**88.2**	84.6

Table 4: Experiments on Ablation Study

Methods	JOBS	GEO
full model(constituency graph)	92.9	88.2
without const tree	90.0	86.8
without word order	87.1	85.4
without separate attention	83.6	77.1
replace separate attention with uniform attention	90.7	87.1
without bilstm	89.3	86.4

Results and Discussions Table 1 and 2 show the comparison results on datasets *JOBS* and *GEO*. We can observe that in general the Graph2Tree, which is proposed in this work, outperforms the Graph2Seq in generating high-quality

Table 5: Output results of "what jobs can a delphi developer find in san antonio on windows ?"

Methods	Prediction results
Reference str	job (ANS) , language (ANS , 'delphi') , title (ANS , 'developer') , loc (ANS , 'san antonio') , platform (ANS , 'windows')
Graph2tree	job (ANS) , language (ANS , 'delphi') , title (ANS , 'developer') , loc (ANS , 'san antonio') , platform (ANS , 'windows')
Graph2seq	job (ANS) , language (ANS , 'delphi') , title (ANS , 'developer') , platform (ANS , 'windows')
Seq2seq	job (ANS) , language (ANS , 'delphi') , title (ANS , 'developer') , loc (ANS , 'san antonio')

logical forms from graph based inputs, no matter what type of graph construction is used.

In terms of graph construction, if the parsing errors, which are introduced by the imperfection of CoreNLP tool, remain in the graph, the performance of both parsers degrades and is even not comparable to that with only *Word Order*. Similarly, the hop size of constituency tree, i.e. the structural complexity, has large impacts of the performance as well. If structural information is overwhelming or little, the performance of parsers drops as well.

On the contrary, when the noises caused by inputs is controllable or reduced by certain approach, the performance of *Word Order + Dependency* could be significantly improved; when the right layers are selected, the performance of *Word Order + Constituency* could also be improved. For example, the logical form accuracies of *Word Order + Constituency* in the one layer cut is higher than *Word Order* respectively. In fact, these accuracies are higher or comparable to other existing parsers achieved, as shown in Table 3. Therefore, the proper graph construction with consideration of noise and structural complexity play an important role in graph based semantic parsers.

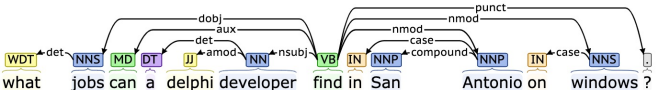

Figure 4: Graph with word sequence and dependency tree

As shown in Table 5, we also present different decoding results from our model and baseline models of sentence *"what jobs can a delphi developerfind in san antonio on windows ?"*. And as shown in Figure 4, we present dependency parsing result of this sentence. We see that the noun key word "jobs" is actually far away form its attribute words "windows" and "san antonio". The traditional sequence decoder is hard to learn a good representation due to the information loss in sequence encoder. But with the syntactic graph input containing dependency information, for example, to the word "jobs", its important relation with word "windows" can be linked in 2 hops. In contrast, if we treat original sequence structure as a line graph, it then needs 10 hops from the word "jobs" to the word "windows". Therefore, syntactic graph with graph encoder is a more effective way to learn a high-quality representation for decoding. And that is part of reasons why our Graph2tree model outperforms the performance of Seq2Seq, Seq2Tree, and the results in table 5 also confirms our analysis. Finally, as shown in Table 4, we perform ablation study on constitution tree based graph. We observe that separate attentions on different parts of nodes (from constituency tree or word tokens) and using constituency tree are two most important factors on the final performance.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a Graph2Tree model consisting of a graph encoder and a hierarchical tree decoder, which could make full use of structured inputs (text) and structured outputs (logic form). We studied the impacts of different syntactic graph constructions on the performance of GNN-based semantic parsers and shed light on how to construct a proper choice of graph for this type of neural semantic parsing.

REFERENCES

[1] David Alvarez-Melis and Tommi S Jaakkola. 2017. Tree-structured decoding with doubly-recurrent neural networks. *ICLR* (2017).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675* (2017).

[4] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. *arXiv preprint arXiv:1806.09835* (2018).

[5] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

[6] Bo Chen, Le Sun, and Xianpei Han. 2018. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. *arXiv preprint arXiv:1809.00773* (2018).

[7] Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280* (2016).

[8] Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793* (2018).

[9] 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*. 315–323.

[10] Jetic Gū, Hassan S Shavarani, and Anoop Sarkar. 2018. Top-down Tree Structured Decoding with Syntactic Connections for Neural Machine Translation and Parsing. *arXiv preprint arXiv:1809.01854* (2018).

[11] Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *arXiv preprint arXiv:1606.03622* (2016).

[12] Zhanming Jie and Wei Lu. 2018. Dependency-based hybrid trees for semantic parsing. *arXiv preprint arXiv:1809.00107* (2018).

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744* (2016).

[16] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[17] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 55–60.

[18] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535* (2017).

[19] 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* 4 (2016), 127–140.

[20] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. *arXiv preprint arXiv:1805.02473* (2018).

[21] 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* (2018).

[22] Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. 2018. Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. *arXiv preprint arXiv:1808.07624* (2018).

[23] Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696* (2017).

[24] Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. 2018. Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. *arXiv preprint arXiv:1806.07832* (2018).

[25] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-DomainText-to-SQL Task. *arXiv preprint arXiv:1810.05237* (2018).

[26] Yuhao Zhang, Peng Qi, and Christopher D Manning. 2018. Graph convolution over pruned dependency trees improves relation extraction. *arXiv preprint arXiv:1809.10185* (2018).