BertGAT: Deep Bidirectional Transformers and Graph Attention Networks for Text-to-SQL task

G058 (s1783039, s1784383, s1829591)

Abstract

Text-to-SQL task is transforming natural language into SQL statements. Generating complex SQL queries with multiple tables and clauses efficiently are still unsolved in cross-domain text-to-SQL tasks. In this paper, we use Spider dataset as our main dataset to fulfill the large-scale and cross-domain semantic parsing text-to-SQL tasks. For this task, we propose BertGAT, which is a novel approach to the before-mentioned task. To build this model, we implement Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) to pre-train deep bidirectional representations instead of the traditional Bidirectional recurrent neural networks. Fine-tuning is applied to the pre-trained BERT representations so that we can use just one extra output layer to create state-of-art models for wide-ranging text-to-SQL tasks. We use Syntax tree network to employ a tree-based SQL generator, and use Graph Attention networks (GATs) (Velickovic et al., 2017) to learn the features of syntax-tree. As far as we know, we are the first to use graph information via implementing BertGAT to compare the syntax dependency tree of questions and its corresponding SQL syntax tree to deal with text-to-SQL tasks. Experimental results show that the BertGAT model can improve the matching accuracy in predicting KEYWORDS, GROUP, HAVING and MULTI-SQL module than the previous state-of-art model.

1 Introduction

Text-to-SQL task is translating natural language queries to interrelated SQL queries, which is a sub-task of semantic parsing in natural language processing.

Recently, some state-of-art methods like attention-enhanced encoder-decoder model presented by (Dong & Lapata, 2016) shows about 80% matching accuracy on JOBS (Tang & Mooney, 2001), GEO (Zettlemoyer & Collins, 2012), ATIS (Elsts et al., 2018), and IFTTT (Quirk et al., 2015) benchmark datasets. Another method TypeSQL proposed by (Yu et al., 2018a), which is a novel approach to view text-to-SQL task as a slot filling task. Specifically, it employs a type of information to grasp scarce entities and numbers in questions. It also gains a high matching accuracy which is 82.6% on the WikiSQL dataset. Then the SQLNet presented by (Xu et al., 2017) is mainly filling in the slots in the sketch to generate structured queries from natural language. It uses seq-to-set and column attention to solving the order issues in the traditional seq-to-seq model, which perform over 90% matching accuracy on WikiSQL dataset. These models seem to have already solved the most problems in the field of text-to-SQL tasks.

However, due to the WikiSQL and those benchmark datasets contain simple SQL queries and database schemas, these model cannot solve nested queries on unseen databases so it performs poorly on the cross-domain dataset Spider. Although the latest method SyntaxSQLNet presented by (Yu et al., 2018c) solves these problems via Syntax tree network, it still generates some errors in column prediction.

Besides, current datasets for Semantic Parsing have some drawbacks. For example, ATIS and Geo datasets used by (Dong & Lapata, 2016), these data sets contain only one database. Most of these databases contain only less than 500 unique SQL queries. Models trained on these data sets are only valid for a particular database. After converting the database, the model will fail ultimately. As for the WikiSQL database, it contains many SQL queries and tables, but all SQL queries are simple and involve only SELECT and WHERE clauses. Also, each database is merely composed of simple tables without foreign keys. Models trained on WikiSQL dataset can still work on other new databases, but the model cannot deal with the complex SQL (such as nested queries or GROUP BY clauses) and multiple table-based databases.

In this paper, we propose a network named BertGAT to fulfill the cross-domain semantic parsing text-to-SQL tasks. We implement Graph Attention networks (GATs) to effectively capture the syntactic information from questions and improve the accuracy of predicting the SQL structures and operators. The graph features among syntax dependency trees of natural language questions and SQL syntax trees, which we captured in this paper is a novel and creative perspective in the field of semantic parsing text-to-SQL tasks. Our contributions are as follows:

- We use large-scale and cross-domain dataset Spider (Yu et al., 2018d) as our main dataset to deal with more complex text-to-SQL tasks than the models built on simple datasets.

- We employ the fine-tuned BERT to embed the natural language questions, which is two-way transformer encoder. Because the decoder is not to be able to predict the information. The main innovation points of the model are in the Pre-train method, that is, the masked Language Model and next sentence prediction, these two methods are used to capture the representation of words and sentence levels, respectively. Additionally, in comparison to RNN, BERT is more efficient and captures longer-distance dependencies. Compared to the previous pre-training model, it captures bidirectional context information in the real sense of the meaning.

- We found out that the syntax dependency tree of questions and its corresponding SQL syntax tree share some similarity. So we apply a powerful feature extractor GAT to extract features from the syntax-trees of question sentence for guiding the decoder. To our knowledge, this is the first time implementing Graph attention networks in dealing with text-to-SQL tasks. The simplicity of GATs and their appropriateness to syntax graph structure can improve the performance of our model.

2 Related Works

Semantic parsing maps a natural-language sentence into a formal representation of meaning representations. There are a range of semantic parsing representations, for example logic forms and Executable Semantic Parsers (Reddy et al., 2016; Long et al., 2016; Basile, 2015; Martin & White, 2011; Liang, 2016)

The text-to-SQL task is a sub-task of semantic parsing which has been extensively studying recently (García et al., 2014; Yu et al., 2018e; Xu et al., 2018b; Shi et al., 2018; Finegan-Dollak et al., 2018b; Yu et al., 2018a;d;b; Xu et al., 2018a; Finegan-Dollak et al., 2018a). In this paper, we tend to focus on graph neural network approaches (Zhu et al., 2019; Fan et al., 2019). (Dong & Lapata, 2016) proposed a seq2seq method to transform texts into logical forms. (Xu et al., 2017) further improve the accuracy on the WikiSQL datasets via the sequence-to-set SQLNet model. (Dong & Lapata, 2018) propose a Coarse-to-Fine Decoding for Neural Semantic Parsing, which get higher accuracies on sevel datasets like WikiSQL.

Our BertGAT model is related to recent work that exploit BERT(Devlin et al., 2018) and GAT (Velickovic et al., 2017) approach. However, this is the first time to combine these two methods and implement it on text-to-SQL tasks.

3 Data set and task

3.1 Dataset

The dataset we used here is the Spider (Yu et al., 2018d), which is a sizeable cross-domain dataset. Spider contains complex SQL queries, like multiple clauses and nested queries. Spider dataset was developed by (Yu et al., 2018d), which is a large-scale human labeled text-to-SQL dataset include 10181 questions, 5693 unique complex SQL queries, and 200 databases with multiple tables. We use this complex dataset and different databases for training and testing so that we need to make our models to generalize to new, unseen databases to deal with complex cross-domain text-to-SQL tasks.

Our data is split as follows:[1] Question split: question examples are split into 8659 train, 1034 dev, 2147 test randomly. [2] Database split: databases are split into 146 train, 20 dev, and 40 tests.

3.2 Task and challenges

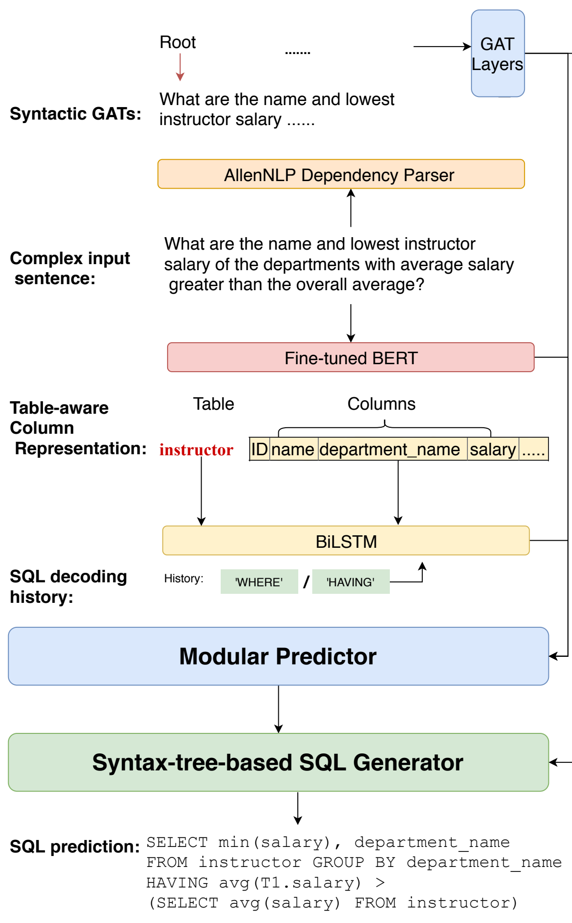
Our task is to create a novel model BertGAT to truly understand the meaning of the natural language questions under different databases. The main idea for building this model is implementing BERT (Devlin et al., 2018) to pre-train deep bidirectional representations. The input here is BERT word representation, which is the sum of the token embeddings, the segmentation embeddings, and the position embeddings. The output of the BERT model is incorporated into Bi-directional LSTM to further generate question embeddings. Besides, our novel model learn features of the structure of syntax dependency tree of questions and its corresponding SQL syntax tree via GAT. The input of the GAT model is questions embeddings and the output is SQL structures. Then we train and test our model based on different SQL queries from different databases. Thus, we can make our model predict SQL tokens on new and unseen databases. Moreover, we predict table names in the FROM clause regarding the relations among tables in the database. We focus on dealing with foreign key constraint problems which have not been solved yet. Furthermore, we improve the comparable accuracy of our model when decoding syntax trees via GATs. Also, we increase the numbers of our training data by data augmentation technique.

We evaluate our model by using SQL Component Matching and Exact Matching proposed by (Yu et al., 2018b). We decompose the predicted queries to SQL clauses to compute the component matching scores. Besides, we evaluate the ground truth and each predicted clause as sub-components to check whether the two sets match accurately. Also, the exact matching score is 1 for those model which can predict all clauses exactly.

In comparison with the existing models, the challenges for our model is two fold. First, to handle complex cross-domain SQL queries in Spider dataset while using graph information to predict SQL structures correctly. Second, to capture relationships among columns and foreign keys in the database.

4 Methodology

Inspired by (Yu et al., 2018b), our model is composed of the similar encoder-decoder model that recursively generates


Root
....
GAT Layers
Syntactic GATs:
What are the name and lowest instructor salary ......
AllenNLP Dependency Parser
Complex input sentence:
What are the name and lowest instructor salary of the departments with average salary greater than the overall average?
Fine-tuned BERT
Table-aware Column Representation:
Table
Columns
instructor
ID name department_name salary .....
BiLSTM
SQL decoding history:
History:
'WHERE'
/
'HAVING'
Modular Predictor
Syntax-tree-based SQL Generator
SQL prediction:

```
SELECT min(salary), department_name
FROM instructor GROUP BY department_name
HAVING avg(T1.salary) >
(SELECT avg(salary) FROM instructor)
```

Figure 1. The overview structure of BertGAT which employs the fine-tuned BERT layer for question representations and syntactic GAT layers for extracting syntax information from questions to guide SQL generation

SQL query based on a SQL syntax tree. However, our model stands different from theirs in two aspects: (1) the fine-tuned BERT is used to encode the natural language questions instead of the bi-directional LSTM, $BiLSTM^{Q}$; (2) the syntactic GATs is applied to extract complimentary syntax features from questions to guide the SQL generation.

4.1 Model Overview

Compared to seq2seq based model for SQL generation, our model is composed of the more complex encoder and the modular decoder for recursively generating SQL queries based on SQL syntax tree. As illustrated in Figure 1, except for using the table-aware column representation and SQL decoding history (similar to the SyntaxNet (Yu et al., 2018b)) , we further employ the fine-tune BERT to learn the representations of questions and use Syntactic GAT to extract the syntax feature to guide the SQL generation. Given the questions in natural language, our model simultaneously takes it as the inputs of syntax dependency parser and fine-tuned BERT for generating their corresponding syntax dependency trees and sentence embeddings respectively. The syntax dependency graphs are further processed by the GAT layers to extract the syntactic information to guide the recursive decoding steps. The decoder includes modular predictor and tree-based SQL generator. Similar to SyntaxNet, the modular predictor contains the following modules:

- **Set operation Module**: predicting `INTERSECT`, `UNION`, `EXCEPT` and `NONE`, which is related to nested queries.
- **COL Module**: predicting table columns in each query.
- **Keyword Module**: predicting keywords including `WHERE`, `GROUP BY` and `ORDER BY`.
- **OP Module**: predicting operators such as =, >, <, etc.
- **AGG Module**: predicting aggregators from `MAX`, `MIN` , `SUM` , `COUNT` , `AVG` and `NONE`.
- **RT Module**: predicting `ROOT` indicating a new nested query or the Terminal value of current query.
- **AOR Module**: predicting `AND` or `OR` operator if there is more than one conditions existed.
- **DAL Module**: predicting the descent, ascent or limit keyword related to `ORDER BY`.
- **HAVING Module**: predicting `HAVING` if the `GROUP BY` has been predicted at early stage.

The modules of modular predictor may at maximum take the input information from 4 sources: the question embedding, the syntactic features, the table-aware column representations, and the SQL decoding history. The tree-based SQL generator employs the stack to recursively generate the SQL query and predict next invoked module according to the SQL grammar.

4.2 Input Encoder

4.2.1 Fine-tuned BERT for Question Representation

As the BERT has been reported to obtain new state-of-the-art performance on several natural language processing tasks (Devlin et al., 2018), we integrate it into our text-to-SQL model to further explore its benefits on this domain-specific task. BERT is a fine-tuning based approach for language representation. To incorporate the bidirectional context, the BERT used the masked language model (MLM), which is based on the Cloze task (Taylor, 1953). The tokens from the input are randomly masked by the masked language model to predict the original word id of the masked word only based on the context. It also further introduces the "next sentence prediction" task, which enables it to handle the text-pair representations. We replace the Glove word embeddings by the embeddings generated by fine-tuned BERT for learning the question representations mainly for two reasons. As the pre-trained representation model, BERT could sharply reduce the engineering effort involved in designing the task-specific architecture for SQL generation. Furthermore, BERT has been proved to achieve state-of-art results on many sentence-level tasks only by fine-tuning.

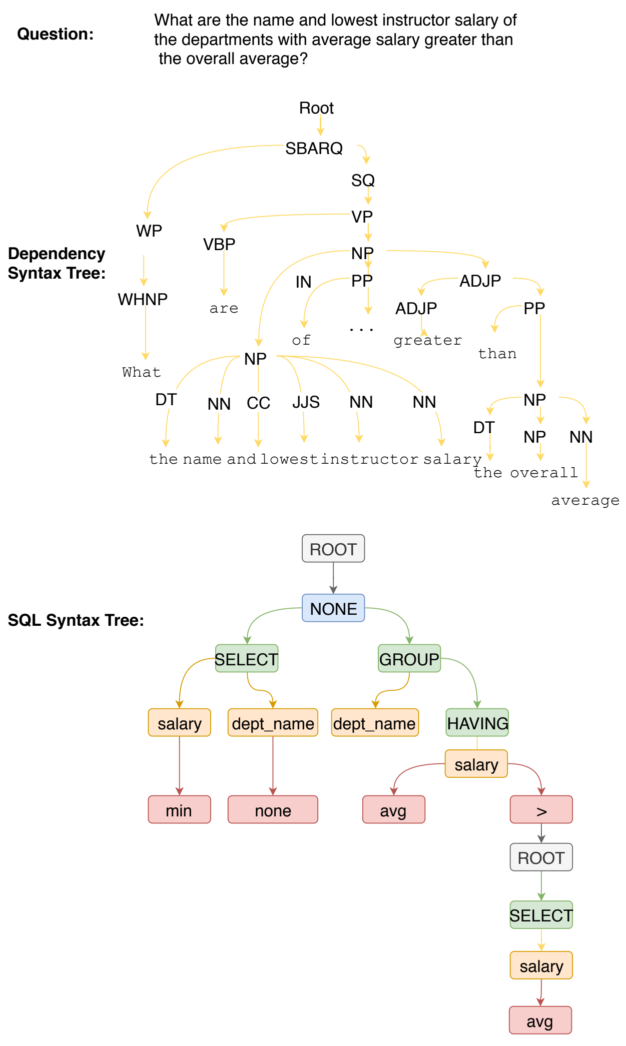


Figure 2. The structures of the syntax dependency tree of questions and its corresponding SQL syntax tree share some similarity

4.2.2 Syntactic Graph Attention Networks

The Graph Convolution Networks (GCNs) was introduced by (Kipf & Welling, 2016), and gradually applied to various natural language tasks, such as text classification and machine translation (Zhou et al., 2018). After that, (Velickovic et al., 2017) presents its self-attentional version called Graph Attention Networks (GATs). By incorporating masked self-attentional layers, GATs can implicitly assign different weights to different neighborhood nodes. Inspired by (Marcheggiani & Titov, 2017), we noticed that previous work did not consider the syntactic information of input questions as the potential benefit for the text-to-SQL tasks. As for encoding sentence with semantic role labeling task, GATs were proved to be able to exploit informative syntactic features and enhance the prediction accuracy. As the syntax trees between SQL queries and natural language questions may share some similarity (Figure 2), we employ the GAT to explicitly extract the syntactic features from the question and use it as the input of modular predictor and tree-based SQL generator to guide the SQL decoding process. For better capturing the informative syntactic features, we further convert the syntax dependency tree into the unidirectional graph.

The GAT layers take a set of node features as input, $\mathbf{E} = \left\{\overrightarrow{h_1}, \overrightarrow{h_2}, ..\overrightarrow{h_N}\right\}, \overrightarrow{h_i} \in \mathbb{R}^F$, where N is the total number of nodes, and F is the number of features given by each node, and compute the output, $\mathbf{E}' = \left\{\overrightarrow{h'_1}, \overrightarrow{h'_2}, ..\overrightarrow{h'_N}\right\}, \overrightarrow{h'_i} \in \mathbb{R}^F$. Compared to normal GCNs, the GATs enable the nodes of a same neighborhood to have different importance by the attention mechanism which could be formulated as:

$$a_{ij} = \frac{exp(\text{LeakyReLU}(\overrightarrow{a}^T[\mathbf{W}\overrightarrow{h_i}][\mathbf{W}\overrightarrow{h_j}]))}{\sum_{k \in N_i} exp(\text{LeakyReLU}(\overrightarrow{a}^T[\mathbf{W}\overrightarrow{h_i}][\mathbf{W}\overrightarrow{h_k}]))} \quad (1)$$

, where a_{ij} annotates the attention score between node i and j, and N_i stands for the total number of nodes.The non-linearity and multi-head attention mechanism will be further apply to obtain the output features of each node:

$$\overrightarrow{h'_i} = \|_{k=1}^{K} \sigma(\sum_{j \in N_i} \alpha_{ij}^k \mathbf{W}^k \overrightarrow{h_j}) \quad (2)$$

where $\|$ stands for the concatenation, α_{ij}^k are the normalized attention coefficients, K stands for the hidden layers of GATs and σ is the sigmoid function.

4.2.3 Column and SQL Decoding History Representation

Similarly to the SyntaxNet, our original input of the column is the list for words in its table name and column name as well as type information of the column, such as date, number, number, primary key, and foreign key (Yu et al., 2018b). Our encoders can capture both the global and local features in the database architecture to understand the given natural language questions.

4.3 Module Prediction

Our model utilizes the sketch-based modular predictor, which incorporate the sequence-to-set prediction approach to get rid of the order issues cause by the sequence-to-sequence model. Similar to SyntaxNet (Yu et al., 2018b), we use different modules designed explicitly for column, operator and keyword prediction, and the information exchange between different module by taking the previous SQL decoding path as extra features. The prediction output of each module is pushed into the stack at each step, and the stack is used for recursive SQL generation at a later stage. As the oppose of SyntaxNet, we use the fine-tuned BERT to learn the representation of input questions and employ syntactic GATs to explicitly capture the informative syntax information from the questions to guide the SQL generation.

The conditional embedding $\mathbf{E}_{a/b}$ of the embedding $\mathbf{E}_a$ on condition to $\mathbf{E}_b$ is computed as:

$$\mathbf{E}_{a/b} = softmax(\mathbf{E}_a \mathbf{W} \mathbf{E}_b^T)\mathbf{E}_a \quad (3)$$

and the distribution of each module is obtained by:

$$P(\mathbf{U}) = softmax(\mathbf{V}tanh(\mathbf{U})) \tag{4}$$

where $\mathbf{U}$ is the output and $\mathbf{V}$ is trainable weights. We use $\mathbf{E}_q$, $\mathbf{E}_{syn}$, $\mathbf{E}_{col}$, and $\mathbf{E}_{hs}$ to represent the output of BERT and hidden state of bi-LSTM on question embedding, syntax features, columns embedding and decoding history comparatively. As there are two different cases for multiple keywords embeddings and single keyword embeddings, we denote them as $\mathbf{E}_{mk}$ and $\mathbf{E}_k$ separately. With these denotations, we partially modify the computational procedure of keyword-related modules from the original SyntaxNet as follows:

- **Multi SQL Module**: the probabilities distribute on keywords `INTERSECT`, `UNION`, `EXCEPT` and `NONE` is calculated by the following equation:

$$P_{IUEN} = P(\mathbf{W}_1\mathbf{E}^T_{q/mk}+\mathbf{W}_2\mathbf{E}^T_{hs/mk}+\mathbf{W}_3\mathbf{E}^T_{syn/mk}+\mathbf{W}_4\mathbf{E}^T_{mk}) \tag{5}$$

- **Keyword Module**: Similarly, we also separately predict the number and value of keyword by two steps:

$$\begin{aligned} P^n_{KW} &= P(\mathbf{W}^n_1\mathbf{E}\mathbf{n}^T_{q/kw} + \mathbf{W}^n_2\mathbf{E}\mathbf{n}^T_{hs/kw}) \\ &\quad +\mathbf{W}^{num}_3\mathbf{E}\mathbf{n}^T_{syn/kw}) \\ P^{val}_{KW} &= P(\mathbf{W}^{val}_1\mathbf{E}\mathbf{val}^T_{q/kw} + \mathbf{W}^{val}_2\mathbf{E}\mathbf{val}^T_{hs/kw} \\ &\quad +\mathbf{W}^{val}_3\mathbf{E}\mathbf{val}^T_{syn/kw} + \mathbf{W}^{val}_4\mathbf{E}\mathbf{val}^T_{kw}) \end{aligned} \tag{6}$$

- **RT Module**: We predict the `ROOT` which starts a new nested query or terminate current query as follows and denote the already predicted column as $\mathbf{E}_{cp}$:

$$P_{ROOT} = P(\mathbf{W}_1\mathbf{E}^T_{q/cp}+\mathbf{W}_2\mathbf{E}^T_{hs/cp}+\mathbf{W}_3\mathbf{E}^T_{syn/cp}+\mathbf{W}_4\mathbf{E}^T_{cp}) \tag{7}$$

- **HAVING Module**: We predict the `HAVING` if the `GROUP BY` has been predicted at early stage as:

$$P_{HAVING} = P(\mathbf{W}_1\mathbf{E}^T_{q/cp}+\mathbf{W}_2\mathbf{E}^T_{hs/cp}+\mathbf{W}_3\mathbf{E}^T_{syn/cp}+\mathbf{W}_4\mathbf{E}^T_{cp}) \tag{8}$$

4.4 Tree-based SQL Generation

The tree-based SQL Generation is a measure to trigger different modules recursively based on the algorithm by (Yu et al., 2018b). We use a stack for decoding and pop one SQL token instance at each step. Then we invoke different modules according to grammar to predict the next token instance and put it into the stack. Repeat the steps above until the stack is empty.

5 Experiments

5.1 Baseline

The preliminary experiments are conducted on TypeSQL, SyntaxNet, and SQLNet, which would later be chosen as the baseline of our proposed models. First, each model is trained on some training data with 300 epochs, where the algorithm is to take 300epoch to save the highest accuracy model on the dev set. Then we test our models using the development data. In this way, we obtained the predicted SQL for each model. Finally, we compared these predicted SQL queries with the gold values and obtained the following results.

	MEDIUM	HARD	EXTRA	ALL
COUNT	440	174	170	1034
EXECUTION ACC(SQLNET)	0.105	0.121	0.029	0.121
EXACT ACC(SQLNET)	0.109	0.086	0.012	0.130
EXECUTION ACC(TYPESQL)	0.107	0.109	0.071	0.119
EXACT ACC(TYPESQL)	0.080	0.052	0.012	0.102
EXECUTION ACC(SYNTAXNET)	0.218	0.264	0.159	0.225
EXACT ACC(SYNTAXNET)	0.230	0.230	0.024	0.248

Table 1. Execution and Exact matching accuracy comparison

PARTIAL MATCHING F1	SQLNET	TYPESQL	SYNTAXNET
SELECT	0.454	0.499	0.658
SELECT(NO AGG)	0.460	0.501	0.672
WHERE	0.231	0.177	0.327
WHERE(NO OP)	0.280	0.229	0.440
GROUP(NO HAVING)	0.473	0.228	0.605
GROUP	0.374	0.149	0.565
ORDER	0.540	0.549	0.643
AND/OR	0.938	0.934	0.940
IUEN	1.00	1.000	0.041
KEYWORDS	0.646	0.620	0.733

Table 2. F1 score comparision

SQLNet is proposed to avoid the "order-matters" problems in a sequence-to-sequence model, and TypeSQL aims at improving SQLNet by reasonably grouping different slots and capturing relationships between attributes. SyntaxNet exploits syntax information for code generation tasks by using sequence-to-set modules for each grammar component and generating SQL syntax trees recursively. When comparing their F1 scores on partial matching, SyntaxNet outperforms other two models whereas TypeSQL and SQLNet give comparable results on most of the metrics.

5.2 GAT Embeddings

We use AllenNLP (Gardner et al., 2018) library in our experiment to generate the syntatic denpendency graph, which is built by the Allen Institute for Artificial Intelligence, for implementing deep learning methods to fulfill the Natural Language Processing task. The model we used here is Dependency Parser from AllenNLP, which analyzes the grammatical structure of a sentence. This model is based on the work of Deep Biaffine Attention for Neural Dependency Parsing(Dozat & Manning, 2016), which is trained on the PTB3.0 dataset via Stanford dependencies. The dependency parser of AllenNLP establish the relationships between "head" words and its corresponding words which modify those heads and achieving 95.57% and 94.44% for unlabeled and labeled attachment score based on gold POS tags. After acquiring the raw dependency tree from the off-shelf dependency parser, we further convert the dependency tree to bi-direction graph for enabling the GAT to attend the impact direction flexibility. The nodes of our GAT are the terms recognized by the AllenNLP dependency parser, and then we use pre-trained BERT model to embed those terms. The embedding produced by pre-trained BERT model act as the features of each node in GATs.

Multi SQL Module	Best Dev Acc
Baseline	0.9678
+ GATs ($\alpha = 1 \times 10^{-4}$)	0.9647
+ GATs ($\alpha = 5 \times 10^{-5}$)	**0.9686**
+ GATs ($\alpha = 1 \times 10^{-5}$)	0.9631
GAT with $\alpha = 5 \times 10^{-5}$	
Hidden Unit=100	**0.9686**
Hidden Unit=200	0.9663
Hidden Unit=400	0.9661

Table 3. The ablation study of GAT extension with Multi SQL Module.

Our GAT model is implemented based on Deep Graph Library (DGL) [1]. DGL is a Python package built for application of graph neural network model group, based on existing deep learning frameworks, like PyTorch, MXNet, Gluon and so on. DGL reduces the implementation of graph neural networks into declaring a set of modules in PyTorch. Besides, DGL provides versatile controls over message passing, ranging from low-level operations to high-level control. Also, it has good scalability to graphs with huge vertices.

The GATs introduce several extra hyper-parameters such as the number of multi-heads of attention, the number of hidden units in GAT convolution layers, and the number of GAT layers. As increasing the number of GAT layers and number of multi-heads of attention would significantly lengthen the training time, we set it to 2 and 4 respectively to make sure the entire training time is affordable. The number of GAT layers N actually controls the scope of graph convolutions. For instance, if the $N = 1$ the GAT can only capture the information from the nearest neighbour nodes, and in our case the two-layer GAT will acquire the semantic information from all the 2-order neighbour terms according to the bi-directional graph built on dependency tree. The number of multi-attention heads decides the context range may be attended. Considering the fact that there are 4 different modules needed to train independently, we firstly grid-search the hyper-parameters including learning rate α in the range $[1 \times 10^{-5}, 1 \times 10^{-4}]$ and the number of hidden unit in range of [100,400] with the `Multi SQL Module`, to narrow down the rough search space of hyper-parameters.

According to the results in Table 3, we would argue that the combination of learning rate $\alpha = 5 \times 10^{-5}$ and the hidden units of 100 could give the optimal performance of GATs extended model. As shown in Figure 3, the extended

[1]https://www.dgl.ai/

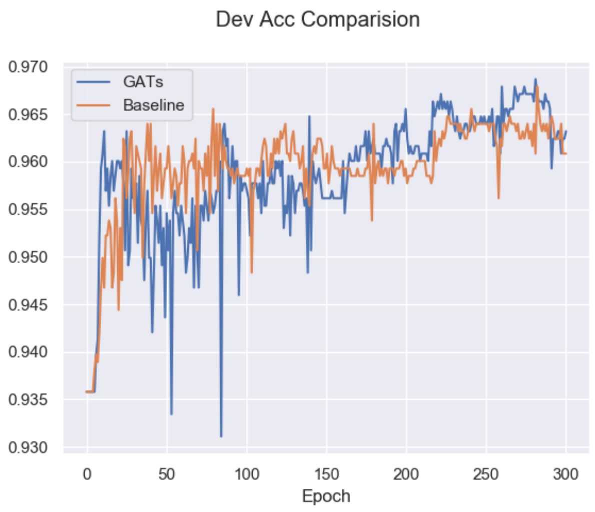


Figure 3. The comparison of dev accuracy between baseline and the extended model with syntax GATs with different parameters.

GAT model outperforms the baseline by increasing the dev accuracy by 0.8%. However, the duration of each training epoch is roughly 3 times longer than the baseline model because all the GATs are dynamic generated according to the input questions.

5.3 BERT Question Embedding

The intuition that why we consider switching from Glove embeddings to Bert embeddings is that Glove is a context-free model, generating a single word embedding representation for each word in the vocabulary. However, when it comes to embedding the question tokens into our model, it is more plausible to use context-dependent models such as Bert since tokens within one sentence often have strong correlation and a better capturing of such dependency might yield more semantically reasonable query expressions. In addition, different questions corresponding to the same SQL query should have higher sentence similarity in the Bert-embedding space than the Glove-embedding space. To testify this proposal, we select all pairs of sentences, each of which corresponds to the same SQL query, and measures their similarity using the Bert embeddings and Glove embeddings respectively. The results are shown in Figure 4.

The results show that Bert embedding perform much better than Glove, with larger mode and much smaller variance. Thus, Bert might be better when conducting experiment.

Originally, the baseline system concatenate the question, history and label embedding in the Glove space. What we do is approximately the same, yet switching all the representation from Glove to Bert item by item. We employ the python package *Bert as a service* (Xiao, 2018), which can covert the sentence to its embedding efficiently with concurrent implementation. We spurned the original implementation of Bert which embeds sentences with dif-

	EASY	MEDIUM	HARD	EXTRA	ALL
SELECT	0.774 → **0.820**	0.557→ 0.554	0.805→**0.810**	0.604→0.596	0.659→**0.664**
SELECT(NO AGG)	0.806 → **0.844**	0.566→ **0.575**	0.816→**0.826**	0.609→**0.629**	0.673→**0.692**
WHERE	0.546 → 0.533	0.363→ **0.385**	0.181→0.167	0.152→**0.181**	0.333→**0.341**
WHERE(NO OP)	0.556 → 0.551	0.464→ 0.460	0.426→0.407	0.291→0.287	0.448→0.439
GROUP(NO HAVING)	0.536 →**0.560**	0.511→ **0.537**	0.698→**0.708**	0.697→0.678	0.593→**0.614**
GROUP	0.500 →**0.533**	0.451→ **0.453**	0.674→0.662	0.684→0.668	0.554→**0.570**
ORDER	0.536 →**0.555**	0.485→ **0.504**	0.684→**0.706**	0.800→**0.802**	0.624→**0.639**
AND/OR	1.000 →0.992	0.916→ 0.912	0.936→**0.964**	0.868→0.859	0.932→0.931
IUEN	0.000 →0.000	0.000→ 0.000	0.077→**0.087**	0.000→**0.062**	0.008→**0.073**
KEYWORDS	0.825 →0.812	0.772→ **0.798**	0.676→**0.680**	0.619→**0.651**	0.733→**0.750**

Table 4. partial matching accuracy comparison: Baseline VS. Bert (Right)

	EASY	MEDIUM	HARD	EXTRA	ALL
SELECT	0.774 → **0.828**	0.557→ 0.553	0.805→**0.827**	0.604→0.604	0.659→**0.677**
SELECT(NO AGG)	0.806 → **0.846**	0.566→ **0.574**	0.816→**0.867**	0.609→**0.632**	0.673→**0.704**
WHERE	0.546 → 0.538	0.363→ **0.394**	0.181→0.163	0.152→**0.190**	0.333→**0.347**
WHERE(NO OP)	0.556 → 0.549	0.464→ 0.457	0.426→0.415	0.291→**0.292**	0.448→0.442
GROUP(NO HAVING)	0.536 →**0.560**	0.511→ **0.537**	0.698→**0.728**	0.697→0.678	0.593→**0.614**
GROUP	0.500 →**0.528**	0.451→ **0.457**	0.674→**0.679**	0.684→0.669	0.554→**0.572**
ORDER	0.536 →**0.555**	0.485→ **0.504**	0.684→**0.706**	0.800→**0.802**	0.624→**0.639**
AND/OR	1.000 →0.992	0.916→ 0.912	0.936→**0.964**	0.868→0.859	0.932→0.931
IUEN	0.000 → 0.000	0.000→ 0.000	0.077→**0.093**	0.000→**0.070**	0.043→**0.081**
KEYWORDS	0.825 →0.815	0.772→ **0.801**	0.676→**0.699**	0.619→**0.654**	0.733→**0.758**

Table 5. partial matching accuracy comparison: Baseline VS. Bert + GAT (Right)

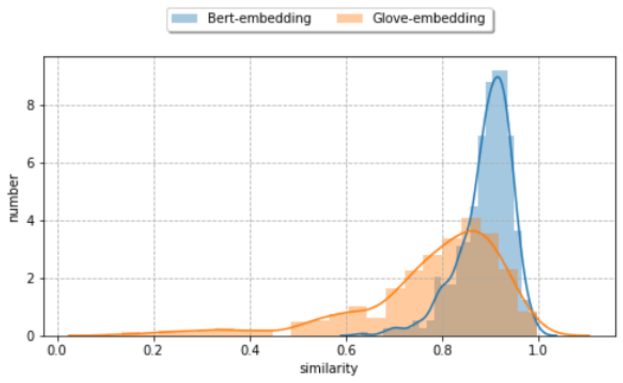


Figure 4. Question similarity comparison: Bert VS. Glove

ference lengths into the same dimension, since it is neither comparable with the Glove nor compatible with our model architecture. Instead, what we do is Elmo-like, process each word in the same sentence the sentence embedding has the same dimension as the sentence length. The comparison with the baseline system is shown in the table 4.

The results show that the Bert embedding performs better than the Glove embedding on almost all the metrics expect *WHERE*. While generally works much better than the baseline model on the **easy** and **medium** modes, the Bert model works only slightly better or even worse on the **hard** mode. This might imply that even Bert cannot handle the complex, multi-level nested text-to-sql questions well. It is quite reasonable why we got a decrease on some of the cases since Bert and Glove embed each sentence into different linear spaces thus it cannot guarantee that one spaces strictly dominates another unless it could be proven somehow. But considering that Bert is trained with a Mask language model and which part of the sentence is masked is random, thus Bert model may be unable to handle some modules well such as *WHERE*, but as mentioned before, this context-dependent model extracts more useful context information and perform better in most of the cases. Thus, we should use Bert instead of Glove when conducting our later text-to-sql experiments.

5.4 Evaluation

Since both the Bert and GAT model gives slightly better results, we might consider combine these two models together to get an overall improvement. For the encoding process, we simply substitute aforementioned Glove embedding with Bert and all the details have been shown in section 5.3. For the decoder part, we use GAT model instead of the original one on the following four modules: **HAVING**, **MULTI-SQL**, **ROOT** and **KEYWORDS**. We choose a smaller learning rate compared to the baseline model in accordance with the GAT's convergence efficiency. The rest of the hyperpameters are the same as what we have discussed in section 5.2. The final comparison with the baseline system is shown in the table 5.

Since GAT only improves the development set accurate to a negligible scale (around 0.01), the overall accuracy are only slighted better than the cases where GAT are not employed in our model. What is interesting is that even though the results are roughly the same as Bert's in the **easy** and **medium** mode, the **hard** mode shows some remarkable improvements on half of the cases.This is in line with our intuition, since it is exactly the four modules we have processed gets some improvements (keywords, having etc). To be more specific, the **KEYWORDS** module increases from 0.680 to 0.699 and the **GROUP** module with or without **HAVING** also shows much difference. What left equally important is the convergence rate when using both GAT and Bert model: it converges much faster than the baseline model. Since Bert also embeds the context information when encoding, such structural information could be transferred to the decoder side and be captured by the GAT model, which has been shown to a better decoder than the baseline. Thus, the decoder could decode the information more easily given the such a structural information, yielding a greater convergence rate.

6 Future Works

There are still some limitations in our model. For example, the running speed of our model is unexpectedly slow. In the future, we plan to accelerate the GAT model by using PyTorch Geometric(PyG) (Fey & Lenssen, 2019). PyG is a library built upon Pytorch (Paszke et al., 2017) for deep learning on erratically structured input data for example graphs and manifolds. Besides, PyG consists of a usable mini-batch loader and a large number of benchmark datasets, which is applicable for arbitrary graphs. PyG is very fast when dealing with sparse data. Compared to the Deep GraphLibrary (DGL) 0.1.3, PyG trains models up to 15 times faster. Thus, we intend to use PyG to enhance the parsing efficiency of our model.

In addition, we only implement 2 GAT layers in our experiment, which allows our model to learn the information within 2-order neighbours around each node. So another work we plan to do in the future is investigating whether the number of layers would boost the performance of our model.

7 Conclusions

In this paper, we presented a BertGAT syntax tree-based model for the text-to-SQL task. We employ Bert as a new word embedding method and use GAT to extract the syntax information and shows an overall improvement on the development set accuracy when combing these two models together. Experimental results show that the BertGAT model improves the accuracy of predicting HAVING, MULTISQL, ROOT and KEYWORDS model. Additionally, it can handle a slightly greater number of complex SQL examples than previous work.

References

Basile, Valerio. *From Logic to Language : Natural Language Generation from Logical Forms. (de la logique à la langue)*. PhD thesis, University of Groningen, Netherlands, 2015. URL https://tel.archives-ouvertes.fr/tel-01342434.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Dong, Li and Lapata, Mirella. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL http://aclweb.org/anthology/P/P16/P16-1004.pdf.

Dong, Li and Lapata, Mirella. Coarse-to-fine decoding for neural semantic parsing. *CoRR*, abs/1805.04793, 2018. URL http://arxiv.org/abs/1805.04793.

Dozat, Timothy and Manning, Christopher D. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734, 2016. URL http://arxiv.org/abs/1611.01734.

Elsts, Atis, Burghardt, Tilo, Byrne, Dallan, Damen, Dima, Fafoutis, Xenofon, Hannuna, Sion L., Ponce-López, Víctor, Masullo, Alessandro, Mirmehdi, Majid, Oikonomou, George, Piechocki, Robert J., Tonkin, Emma, Vafeas, Antonis, Woznowski, Przemyslaw, and Craddock, Ian. A guide to the SPHERE 100 homes study dataset. *CoRR*, abs/1805.11907, 2018. URL http://arxiv.org/abs/1805.11907.

Fan, Wenqi, Ma, Yao, Li, Qing, He, Yuan, Zhao, Yihong Eric, Tang, Jiliang, and Yin, Dawei. Graph neural networks for social recommendation. *CoRR*, abs/1902.07243, 2019. URL http://arxiv.org/abs/1902.07243.

Fey, Matthias and Lenssen, Jan Eric. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Finegan-Dollak, Catherine, Kummerfeld, Jonathan K., Zhang, Li, Ramanathan, Karthik, Sadasivam, Sesh, Zhang, Rui, and Radev, Dragomir R. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 351–360, 2018a. URL https://aclanthology.info/papers/P18-1033/p18-1033.

Finegan-Dollak, Catherine, Kummerfeld, Jonathan K., Zhang, Li, Ramanathan, Karthik, Sadasivam, Sesh, Zhang, Rui, and Radev, Dragomir R. Improving text-to-sql evaluation methodology. *CoRR*, abs/1806.09029, 2018b. URL http://arxiv.org/abs/1806.09029.

García, Jokin, Díaz, Oscar, and Cabot, Jordi. An adapter-based approach to co-evolve generated SQL in model-to-text transformations. In *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, pp. 518–532, 2014. doi: 10.1007/978-3-319-

07881-6_35. URL https://doi.org/10.1007/978-3-319-07881-6_35.

Gardner, Matt, Grus, Joel, Neumann, Mark, Tafjord, Oyvind, Dasigi, Pradeep, Liu, Nelson F., Peters, Matthew E., Schmitz, Michael, and Zettlemoyer, Luke. Allennlp: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640, 2018. URL http://arxiv.org/abs/1803.07640.

Kipf, Thomas N. and Welling, Max. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

Liang, Percy. Learning executable semantic parsers for natural language understanding. *Commun. ACM*, 59 (9):68–76, 2016. doi: 10.1145/2866568. URL https://doi.org/10.1145/2866568.

Long, Reginald, Pasupat, Panupong, and Liang, Percy. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL http://aclweb.org/anthology/P/P16/P16-1138.pdf.

Marcheggiani, Diego and Titov, Ivan. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pp. 1506–1515, 2017. URL https://aclanthology.info/papers/D17-1159/d17-1159.

Martin, Scott and White, Michael. Creating disjunctive logical forms from aligned sentences for grammar-based paraphrase generation. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation@ACL, Portland, Oregon, USA, June 24, 2011*, pp. 74–83, 2011. URL https://aclanthology.info/papers/W11-1609/w11-1609.

Paszke, Adam, Gross, Sam, Chintala, Soumith, and Chanan, Gregory. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, 2017.

Quirk, Chris, Mooney, Raymond J., and Galley, Michel. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 878–888, 2015. URL http://aclweb.org/anthology/P/P15/P15-1085.pdf.

Reddy, Siva, Täckström, Oscar, Collins, Michael, Kwiatkowski, Tom, Das, Dipanjan, Steedman, Mark, and Lapata, Mirella. Transforming dependency structures to logical forms for semantic parsing. *TACL*, 4: 127–140, 2016. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/807.

Shi, Tianze, Tatwawadi, Kedar, Chakrabarti, Kaushik, Mao, Yi, Polozov, Oleksandr, and Chen, Weizhu. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *CoRR*, abs/1809.05054, 2018. URL http://arxiv.org/abs/1809.05054.

Tang, Lappoon R. and Mooney, Raymond J. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, pp. 466–477, 2001. doi: 10.1007/3-540-44795-4_40. URL https://doi.org/10.1007/3-540-44795-4_40.

Taylor, Wilson L. âĂIJcloze procedureâĂİ: A new tool for measuring readability. *Journalism Bulletin*, 30(4): 415–433, 1953.

Velickovic, Petar, Cucurull, Guillem, Casanova, Arantxa, Romero, Adriana, Liò, Pietro, and Bengio, Yoshua. Graph attention networks. *CoRR*, abs/1710.10903, 2017. URL http://arxiv.org/abs/1710.10903.

Xiao, Han. bert-as-service. https://github.com/hanxiao/bert-as-service, 2018.

Xu, Kun, Wu, Lingfei, Wang, Zhiguo, Feng, Yansong, and Sheinin, Vadim. Sql-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 931–936, 2018a. URL https://aclanthology.info/papers/D18-1112/d18-1112.

Xu, Kun, Wu, Lingfei, Wang, Zhiguo, Yu, Mo, Chen, Liwei, and Sheinin, Vadim. Sql-to-text generation with graph-to-sequence model. *CoRR*, abs/1809.05255, 2018b. URL http://arxiv.org/abs/1809.05255.

Xu, Xiaojun, Liu, Chang, and Song, Dawn. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017. URL http://arxiv.org/abs/1711.04436.

Yu, Tao, Li, Zifan, Zhang, Zilin, Zhang, Rui, and Radev, Dragomir R. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pp. 588–594, 2018a. URL https://aclanthology.info/papers/N18-2093/n18-2093.

Yu, Tao, Yasunaga, Michihiro, Yang, Kai, Zhang, Rui, Wang, Dongxu, Li, Zifan, and Radev, Dragomir R. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 1653–1663, 2018b. URL https://aclanthology.info/papers/D18-1193/d18-1193.

Yu, Tao, Yasunaga, Michihiro, Yang, Kai, Zhang, Rui, Wang, Dongxu, Li, Zifan, and Radev, Dragomir R. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *CoRR*, abs/1810.05237, 2018c. URL http://arxiv.org/abs/1810.05237.

Yu, Tao, Zhang, Rui, Yang, Kai, Yasunaga, Michihiro, Wang, Dongxu, Li, Zifan, Ma, James, Li, Irene, Yao, Qingning, Roman, Shanelle, Zhang, Zilin, and Radev, Dragomir R. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 3911–3921, 2018d. URL https://aclanthology.info/papers/D18-1425/d18-1425.

Yu, Tao, Zhang, Rui, Yang, Kai, Yasunaga, Michihiro, Wang, Dongxu, Li, Zifan, Ma, James, Li, Irene, Yao, Qingning, Roman, Shanelle, Zhang, Zilin, and Radev, Dragomir R. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887, 2018e. URL http://arxiv.org/abs/1809.08887.

Zettlemoyer, Luke S. and Collins, Michael. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *CoRR*, abs/1207.1420, 2012. URL http://arxiv.org/abs/1207.1420.

Zhou, Jie, Cui, Ganqu, Zhang, Zhengyan, Yang, Cheng, Liu, Zhiyuan, and Sun, Maosong. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Zhu, Hao, Lin, Yankai, Liu, Zhiyuan, Fu, Jie, Chua, Tat-Seng, and Sun, Maosong. Graph neural networks with generated parameters for relation extraction. *CoRR*, abs/1902.00756, 2019. URL http://arxiv.org/abs/1902.00756.