Graph Enhanced Cross-Domain Text-to-SQL Generation

Siyu Huo
IBM Research, USA
siyu.huo@ibm.com

Tengfei Ma
IBM Research AI, USA
tengfei.ma1@ibm.com

Jie Chen
MIT-IBM Watson AI Lab, USA
chenjie@us.ibm.com

Maria Chang
IBM Research AI, USA
maria.chang@ibm.com

Lingfei Wu
IBM Research AI, USA
wuli@us.ibm.com

Michael Witbrock
University of Auckland
m.witbrock@auckland.ac.nz

Abstract

Semantic parsing is a fundamental problem in natural language understanding, as it involves the mapping of natural language to structured forms such as executable queries or logic-like knowledge representations. Existing deep learning approaches for semantic parsing have shown promise on a variety of benchmark data sets, particularly on text-to-SQL parsing. However, most text-to-SQL parsers do not generalize to unseen data sets in different domains. In this paper, we propose a new cross-domain learning scheme to perform text-to-SQL translation and demonstrate its use on Spider, a large-scale cross-domain text-to-SQL data set. We improve upon a state-of-the-art Spider model, SyntaxSQLNet, by constructing a graph of column names for all databases and using graph neural networks to compute their embeddings. The resulting embeddings offer better cross-domain representations and SQL queries, as evidenced by substantial improvement on the Spider data set compared to SyntaxSQLNet.

1 Introduction

Text-to-SQL translation is currently one of the most important tasks in semantic parsing. It involves mapping natural language sentences to SQL queries that can be executed on associated database tables. Most text-to-SQL data sets have two very important limitations: (1) they mostly contain only very simple SQL queries, and (2) they use the same databases (and often identical SQL queries) for training and testing. Finegan-Dollak et al. (2018) demonstrated that text-to-SQL parsers that perform well on existing benchmarks do not generalize well to unseen SQL queries, as measured by experiments on modified data sets where no two identical SQL queries appear in both the training and testing sets. The Spider data set (Yu et al., 2018c) was developed to address these limitations by including a large number of complex programs, databases with multiple tables, and by ensuring that the SQL queries and databases that appear in the training set do not also appear in the testing set. In this way, Spider can be used as a better measure of a model's ability to produce unseen complex programs and to generalize to new domains.

The Spider data set has led to the development of complex, cross-domain semantic parsers, such as SyntaxSQLNet (Yu et al., 2018b). In prior work, cross-domain semantic parsing referred to the ability to generalize among different logical forms (Su and Yan, 2017). For the text-to-SQL task, however, cross-domain semantic parsing refers to the ability to generalize across different queries and databases. Very recently, SyntaxSQLNet (Yu et al., 2018b) was proposed to solve this task by introducing a SQL-specific syntax tree-based decoder, with a SQL generation path history and table-aware column attention encoder. For better cross-domain performance, it employs a data augmentation method to generate more diverse training examples across databases. However, creating this augmented data is expensive and time-consuming. It requires grouping the SQL patterns and then manually editing the selected SQL patterns and their corresponding lists of questions.

In this paper, we propose a new approach to enhance cross-domain learning in text-to-SQL systems. In most existing systems, selecting entries from available databases is one of the most important components. When experimenting with baselines, Yu et al. (2018c) observed that most of the component matching errors (that is, components of the predicted SQL query do not match those of the gold query) were caused by errors in column prediction. A good representation of the columns in the databases should lead to accurate matching between queries and columns. Our idea is to connect all tables and databases across domains via shared column names, so that the column name representation can use information across domains for better generalization. We implement this idea by constructing a database graph and using a graph neural network to encode the columns. In this way, we obtain higher quality column embeddings, which lead to more accurate column matching and better SQL generation.

2 Related Work

Many semantic parsers have been developed to translate natural language text into structured, symbolic forms, including abstract meaning representation (Lyu and Titov, 2018), executable programs (e.g. Python, Lisp, Bash) (Allamanis et al., 2015; Rabinovich et al., 2017; Yin and Neubig, 2017; Liang et al., 2017; Lin et al., 2018), and SQL queries (Dong and Lapata, 2018; Yu et al., 2018b,a; Xu et al., 2017).

For text-to-SQL parsing, the work most closely related to ours is SyntaxSQLNet (Yu et al., 2018b), which is the state-of-the-art approach for the Spider data set (Yu et al., 2018c). SyntaxSQLNet extends prior text-to-SQL models, such as SQLNet (Xu et al., 2017) and TypeSQL (Yu et al., 2018a), by encoding both local information from column names and global information from table names. The primary difference between SyntaxSQLNet and our work is that we use a novel column embedding technique that additionally includes a graph of the tables, connected through shared column names.

3 Problem Formulation

We focus on the cross-domain SQL generation task within the Spider data set (Yu et al., 2018c). Spider consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains.

Specifically, in the data set each natural language query Q is associated with a corresponding SQL query S and a database DB. The database contains multiple tables T, and each table contains multiple columns C. The task of SQL generation is to generate S given only Q and DB. For cross-domain learning, the databases for training and testing are separate, so that one can test the generalization ability of the models to unseen domains.

4 Method

Our model is based on the framework of SyntaxSQLNet (Yu et al., 2018b) but extends it with our approach for generating column embeddings. We first briefly introduce the SyntaxSQLNet system, and then present the proposed graph-based column embedding method.

4.1 Background of SyntaxSQLNet

For the SQL generation task, one is given a natural language sentence and a database and is asked to generate the corresponding SQL query. Therefore, one may use an encoder to encode the sentence and table columns in the database and use a decoder to generate the SQL query. In SyntaxSQLNet, the sentence is encoded by a bi-directional LSTM (BiLSTM), while each column in the table is encoded by using a simple scheme called table-aware column representation. The scheme takes the list of words in the table name and the column name, as well as the type information of the column, as input to a BiLSTM and outputs the final state as the column representation. This approach incorporates both the global table information and the local column information.

To better leverage SQL structures, SyntaxSQLNet uses an SQL specific tree-based decoder with SQL path histories. It decomposes the decoder into a set of recursive modules: IUEN, KW, COL, OP, AGG, Root/Terminal, AND/OR, DESC/ASC/LIMIT, and HAVING. Each module deals with different SQL components. For example, the KW module predicts keywords from **WHERE**, **GROUP BY** and **ORDER BY**, and the COL module predicts columns. For details of the other modules, see (Yu et al., 2018b). In this work, we focus on improving the COL module.

The recursive modules in SyntaxSQLNet are combined to form the whole generation process. Specifically, when the decoder generates the SQL query, it first determines which module to invoke, and then uses that module to predict the next SQL token. For each module, the input encoding goes through an attention computation, before being used to make the prediction of the next token. For example, in the COL module, the prediction is computed by the using following equations:

Table 1: Example Databases

Database	Table Name	Column Names		
department store	customers	customer id	customer name	customer address
	customer orders	order id	customer id	order date
coffee shop	shop	shop id	address	num of staff

$$P_{\mathrm{COL}}^{\mathrm{num}} = \mathcal{P}\left(\mathbf{W}_1^{\mathrm{num}} \mathbf{H}_{\mathrm{Q/COL}}^{\mathrm{num}}{}^{\top} + \mathbf{W}_2^{\mathrm{num}} \mathbf{H}_{\mathrm{HS/COL}}^{\mathrm{num}}{}^{\top}\right)$$

$$P_{\mathrm{COL}}^{\mathrm{val}} = \mathcal{P}\left(\mathbf{W}_1^{\mathrm{val}} \mathbf{H}_{\mathrm{Q/COL}}^{\mathrm{val}}{}^{\top} + \mathbf{W}_2^{\mathrm{val}} \mathbf{H}_{\mathrm{HS/COL}}^{\mathrm{val}}{}^{\top} + \mathbf{W}_3^{\mathrm{val}} \mathbf{H}_{\mathrm{COL}}{}^{\top}\right),$$

where “num” means the number of columns, “val” means the index(es) of the column(s), “Q” means question, “HS” means path history, “COL” means column, $\mathcal{P}(\mathbf{U}) = \mathrm{softmax}(\mathbf{V}\tanh(\mathbf{U}))$ is a probability distribution given score $\mathbf{U}$ and parameter $\mathbf{V}$, the $\mathbf{W}$'s are learnable parameters, and the $\mathbf{H}_{1/2}$'s are conditional embeddings defined as $\mathbf{H}_{1/2} = \mathrm{softmax}(\mathbf{H}_1 \mathbf{W} \mathbf{H}_2^{\top})\mathbf{H}_1$.

4.2 Database Graph Construction

4.2.1 Motivation

For a good accuracy of the final SQL generation, every module needs to perform well. However, the COL module is a significant bottleneck of the system. When reproducing the results from SyntaxSQLNet, we find that its accuracy is only slightly above 50%, while other modules generally achieve 90%. SyntaxSQLNet gains generalizability across domains through encoding both global and local information by simply using all the words from the table name, column name, and column data type. There are two problems with this approach. First, no explicit ordering exists for these words; hence, the use of BiLSTM to encode them seems less justified. Second, although it adds the global information from table names for better column name embedding, the table names are completely independent of other tables and databases, and therefore it incorporates little information from other domains.

Our idea is to use a graph to connect the column names across all tables and databases and compute representations of them by using a graph neural network. In this way, the information of different domains is passed to each other, so that one can learn a better column embedding that generalizes across domains.

Our approach is relevant to those that use neural networks for domain adaptation; see, e.g., Pareja et al. (2019) who adapt the graph neural network model for data at different time steps. However, our approach is conceptually different from these methods, because we do not map the tasks to a new domain but rather, learn a better representation of the table elements through leveraging all domain information. Another related work that also uses GNN for sementic parsing is the very recently published Bogin et al. (2019), but the graph therein is an abstraction of the database schema, as opposed to being a tool for producing column representations as in our work.

4.2.2 Graph Construction

We construct a graph to connect all column names across tables and databases. To encode more information, we also include the table names and column data types as additional nodes in the graph. In order to obtain more connections and reduce unseen phrases, we split the names into words, so that different tables and databases share more nodes. Specifically, our database graph is constructed as follows:

1. Every column name is separated into a set of words and the words are used as nodes in the graph.

2. Within each table, we connect all word nodes to each other.

3. For each table, we include the words of table name as additional nodes, and connect them with all column name words in that table.

4. Different tables and databases are thus connected through shared words.

5. We also add column data types as additional nodes and connect them with corresponding column name words.

In Table 1 and Figure 1 respectively, we show an example of the databases and the graph con-

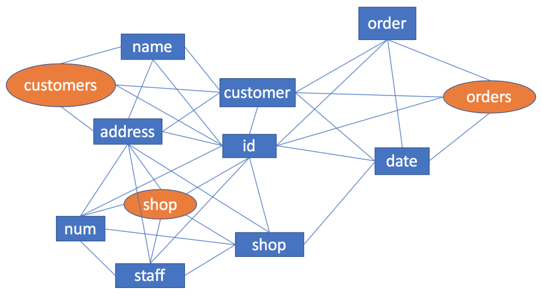

Figure 1: An example of the constructed graph for databases in Table 1. Orange nodes are words in the table names and blue nodes are words in the column names. For simplicity, column data types are not included in this example.

structed for them. Clearly, in the graph, two different databases are connected through shared words, such as id and address.

4.2.3 Graph Encoding

We encode the nodes in the graph by using a 2-layer graph convolutional network (GCN) (Kipf and Welling, 2016). After we obtain the node embeddings for all nodes, for every table column we combine the embeddings of the words of the corresponding table name, column name, and column data type to get the overall representation of the column. We call this approach subgraph pooling. In details, assume that we have a table name TN, column name words C_1, C_2, C_3, and column data type TP. We first use the pre-trained GloVe vectors (Pennington et al., 2014) to initialize the embedding of each node: $E_{TN}, E_{C_1}, E_{C_2}, E_{C_3}, E_{TP}$. With the graph convolutional network, we obtain corresponding embedding vectors $G_{TN}, G_{C_1}, G_{C_2}, G_{C_3}, G_{TP}$. For subgraph pooling, we first concatenate the original GloVe vectors and the newly learned embedding vectors, then do averaging over all nodes for this column, and finally map it to another lower-dimensional space: $H(*) = W_h[\text{Mean}_\omega(E(\omega)||G(\omega))]$, where $||$ denotes concatenation, $\omega \in TN, C_1, C_2, C_3, TP$ and W_h is a learnable parameter. Afterwards, we replace the BiLSTM encoding in SyntaxSQLNet by $H(*)$ and continue the SyntaxSQLNet decoding process.

5 Experiments

5.1 Experimental Setting

We use the Spider data set and follow the setting of Yu et al. (2018c). The data set is split into 7,000/1,034/2,147 train/development/test examples, and the databases are correspondingly split into 146/40/20 for train/development/test. The models are evaluated by exact matching accuracy with the provided test script Yu et al. (2018c). Since we made no changes to the modules other than COL, we do not compare the component matching accuracy as in Yu et al. (2018b). Instead, we include the accuracy of the COL module as an additional metric to better interpret the effects of the proposed graph encoding method.

For GCN (Kipf and Welling, 2016), we used the inductive version. That is, in the training phase we only construct the graph from the training databases, but for testing we connect the test databases with the training ones and use the learned GCN parameters for embedding computation.

5.2 Results

Method	Matching acc.	COL acc.
Seq2seq+attention	1.8%	NA
SQLNet	10.9%	NA
SyntaxSQLNet	18.9%	53.5%
SyntaxSQLNet + Data Augmentation	24.8%	61.9%
GNN-SQL	22.0%	57.0%

Table 2: Comparison of different methods with respect to final exact matching accuracy and COL module accuracy on development set. The results of SyntaxSQLNet were run by ourselves using the provided system.

From the table we see that the COL module accuracy is 3.5% higher than that in SyntaxSQLNet, without performing any modification to other modules. The final accuracy is also 3.1% higher than SyntaxSQLNet without data augmentation. Although our final matching accuracy is lower than the data augmented SyntaxSQLNet (22.0% vs 24.8%), we have demonstrated the usefulness of the improved COL module and the potential for reducing the need and effort of labeling additional augmented data.

6 Conclusion

In this paper we propose a new graph-based method for cross-domain text-to-SQL generation. As opposed to data augmentation in SyntaxSQLNet, we construct a graph to connect all columns across tables and databases, yielding better generalizablity of the column representation and SQL generation. Experimental results on Spider demonstrates the superiority of our method over SyntaxSQLNet without data augmentation.

References

Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *International Conference on Machine Learning*, pages 2123–2132.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *ACL*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

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

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 23–33. Association for Computational Linguistics (ACL).

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E. Leiserson. 2019. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. Preprint arXiv:1902.10191.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

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.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. 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, Volume 2 (Short Papers)*, pages 588–594.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. 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*, pages 1653–1663.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018c. 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*, pages 3911–3921.