Recursive and Clause-Wise Decoding for Complex and Cross-Domain Text-to-SQL Generation

Dongjun Lee
SAP Labs Korea
dongjun.lee01@sap.com

Abstract

Most deep learning approaches for text-to-SQL generation are limited to the WikiSQL dataset, which only supports very simple queries over a single table. We focus on the Spider dataset, a complex and cross-domain text-to-SQL task, which includes complex queries over multiple tables. In this paper, we propose a SQL clause-wise decoding neural architecture with a self-attention based database schema encoder to address Spider task. Each of the clause-specific decoders consists of a set of sub-modules, which is defined by the syntax of each clause. Additionally, our model works recursively to support nested queries. The experimental result shows that our model outperforms the previous state-of-the-art model by 9.8% in the exact matching accuracy on the Spider dev dataset. In addition, we show that our model is significantly more effective to predict complex and nested queries than previous works.

1 Introduction

Text-to-SQL generation is a task of translating a natural language question into the corresponding SQL. Recently, various deep learning approaches have been proposed based on the WikiSQL dataset (Zhong et al., 2017). However, since WikiSQL contains only very simple queries over just a single table, these approaches (Xu et al., 2017; Huang et al., 2018; Yu et al., 2018a; Dong and Lapata, 2018) cannot be applied directly to generate complex queries containing elements such as JOIN, GROUP BY, and nested queries.

To overcome this limitation, Yu et al. (2018c) introduced Spider, a new complex and cross-domain text-to-SQL dataset. It contains a large number of complex queries over different databases with multiple tables. Also, it requires a model to generalize to unseen database schema as different databases are used for training and testing. Therefore, a model should understand not only the natural language question but also the schema of the corresponding database to generate the correct SQL query.

In this paper, we propose a SQL-specific clause-wise decoding neural network model to address Spider task. We first generate a sketch for each SQL clause (SELECT, WHERE, and etc) by text classification modules. Then, clause-specific decoders find the columns and corresponding operators based on the sketches. Our contributions are summarized as follows.

• We decompose the SQL decoding process clause-wisely. We also modularize each of the clause-specific decoders into sub-modules based on the syntax of each clause. Our architecture enables the model to learn clause-dependent context and also ensures the syntactic correctness of the predicted SQL output.

• Our model works recursively so that it can predict nested queries.

• We also introduce self-attention based database schema encoder that enables our model to generalize to unseen databases.

In the experiment on the dev dataset of Spider, we achieve 28.8% exact SQL matching accuracy, which outperforms the previous state-of-the-art approach (Yu et al., 2018b) by 9.8%. In addition, we show that our approach is significantly more effective compared to previous works in generating not only simple SQL queries, but also complex and nested queries.

2 Related Work

Our work is related to the grammar-based constrained decoding approaches for the code gener-
ation task. For example, Yin and Neubig (2017) introduced a grammar model based on the Abstract Syntax Tree (AST) to define the sequence of actions in the decoding process. Similarly, Rabinovich et al. (2017) proposed modular decoders that each module constructs a node in ASTs. However, a text-to-SQL task differs from the code generation in two aspects. First, it takes a database schema as an input in addition to natural language. To predict SQL correctly, a model should fully understand the relationship between the question and the schema. Second, as SQL is a non-procedural language, predictions of SQL clauses do not need to be done sequentially.

For the text-to-SQL generation, several SQL-specific approaches have been proposed (Zhong et al., 2017; Xu et al., 2017; Huang et al., 2018; Yu et al., 2018a; Dong and Lapata, 2018; Yavuz et al., 2018) based on WikiSQL dataset (Zhong et al., 2017). However, all of them are limited to the WikiSQL specific SQL sketch, which only supports very simple queries. It includes only SELECT and WHERE clause, only a single expression in SELECT clause, and works only for a single table. Iyer et al. (2017); Finegan-Dollak et al. (2018) proposed sequence-to-sequence approaches to predict more complex SQL queries. However, they focused just on the specific databases such as ATIS (Price, 1990) and GeoQuery (Zelle and Mooney, 1996). Since they only consider question and SQL pairs without understanding database schema, their approaches cannot generalize to unseen databases.

SyntaxSQLNet (Yu et al., 2018b) is the first and state-of-the-art model for the Spider (Yu et al., 2018c), a complex and cross-domain text-to-SQL task. They proposed a SQL specific syntax tree-based decoder with SQL generation history. They developed 9 modules for different SQL components and generated SQL tokens sequentially by calling one of the modules based on the SQL grammar. Our approach differs from SyntaxSQLNet in the following aspects. First, we separate our modules SQL clause-wisely unlike Yu et al. (2018b) separated modules just by SQL components. For example, in SyntaxSQLNet, a single column prediction module works both in SELECT and WHERE clause, depending on the SQL decoding history. In contrast, we define and train decoding modules separately for each SQL clause to fully utilize clause-dependent context.

3 Methodology

We generate a complex SQL clause-wisely as described in Figure 1. Each clause is predicted consecutively by at most three different types of modules (sketch, column, operator). The same architecture recursively predicts nested queries with temporal generated SQL as an additional input.

3.1 Question and Schema Encoding

We encode a natural language question with bi-directional LSTM. We denote $H_Q \in \mathbb{R}^{d \times |X|}$ as the question encoding, where $d$ is the number of LSTM units and $|X|$ is the number of tokens in the question.

To encode a database schema, we consider each column as a concatenated sequence of words in the table name and column name with separation token. (ex. [student, [SEP], first, name]). First, we apply bi-directional LSTM over this sequence for each column. Then, we apply self-attention mechanism (Lin et al., 2017) over the LSTM outputs to form a summarized fixed-size vector. For $i$th column, its encoding $h^{(i)}_{col} \in \mathbb{R}^{d \times |L|}$ is computed by a weighted sum of LSTM output $o^{(i)}_{col} \in \mathbb{R}^{d \times |L|}$ as follows:

$$\alpha = \text{softmax}(w^T \tanh(o^{(i)}_{col}))$$ (1)

$$h^{(i)}_{col} = o^{(i)}_{col} \alpha^T$$ (2)

where $|L|$ is the number of tokens in the column and $w \in \mathbb{R}^d$ is a trainable parameter. We denote $H_{col} = [h^{(1)}_{col}, ... h^{(|C|)}_{col}]$ as columns encod-
ings where $|C|$ is the number of columns in the database.

### 3.2 Sketch Generation

We generate clause-wise sketch via 8 different text classification modules that include the number of SQL expressions in each clause, presence of LIMIT, and presence of INTERSECT/UNION/EXCEPT as described in Figure 1. All of them share the same model architecture but trained separately. For the classification, we applied attention-based bi-directional LSTM following Zhou et al. (2016).

First, we compute sentence representation $r_s \in \mathbb{R}^d$ by a weighted sum of question encoding $H_Q \in \mathbb{R}^{d \times |X|}$. Then we apply softmax classifier to choose the sketch as follows:

$$
\alpha_s = \text{softmax}(w_s^T \tanh(H_Q)) \quad (3)
$$

$$
r_s = H_Q \alpha_s^T \quad (4)
$$

$$
P_{\text{sketch}} = \text{softmax}(W_s r_s + b_s) \quad (5)
$$

where $w_s \in \mathbb{R}^d$, $W_s \in \mathbb{R}^{n_s \times d}$, $b_s \in \mathbb{R}^{n_s}$ are trainable parameters and $n_s$ is the number of possible sketches.

### 3.3 Columns and Operators Prediction

To predict columns and operators, we use LSTM decoder with attention mechanism (Luong et al., 2015) that the number of decoding steps are decided by the sketch generation module. We train 5 different column prediction modules separately for each SQL clause, but they share the same architecture.

In the column prediction module, the output vector of the decoder at the $t$-th decoding step is computed as $d_{\text{col}}^{(t)}(\in \mathbb{R}^d) = \text{LSTM}(d_{\text{col}}^{(t-1)}, h_{\text{col}}^{(t-1)})$, where $h_{\text{col}}^{(t-1)} \in \mathbb{R}^d$ is an encoding of the predicted column in the previous decoding step. The context vector $r^{(t)}$ is computed by a weighted sum of question encodings $H_Q \in \mathbb{R}^{d \times |X|}$ based on attention weight as follows:

$$
\alpha^{(t)} = \text{softmax}(d_{\text{col}}^{(t)}^T H_Q) \quad (6)
$$

$$
r^{(t)} = H_Q \alpha^{(t)}^T \quad (7)
$$

Then, attentional output of the $t$-th decoding step $a_{col}^{(t)}$ is computed as a linear combination of $d_{col}^{(t)} \in \mathbb{R}^d$ and $r^{(t)} \in \mathbb{R}^d$ following $\tanh$ activation.

$$
a_{\text{col}}^{(t)} = \tanh(W_1 d_{\text{col}}^{(t)} + W_2 r^{(t)}) \quad (8)
$$

where $W_1, W_2 \in \mathbb{R}^{d \times d}$ are trainable parameters. Finally, the probability for each column at the $t$-th decoding step is computed as a dot product between $a_{\text{col}}^{(t)} \in \mathbb{R}^d$ and encoding of each column in $H_{\text{col}} \in \mathbb{R}^{d \times |C|}$ followed by softmax.

$$
P_{\text{col}}^{(t)} = \text{softmax}(a_{\text{col}}^{(t)T} H_{\text{col}}) \quad (9)
$$

To predict corresponding operators for each predicted column, we use a decoder of the same architecture as in the column prediction module. The only difference is that a decoder input at the $t$-th decoding step is an encoding of the $t$-th predicted column from the column prediction module.

$$
d_{\text{op}}^{(t)} = \text{LSTM}(d_{\text{op}}^{(t-1)}, h_{\text{col}}^{(t)}) \quad (10)
$$

Attentional output $a_{\text{op}}^{(t)} \in \mathbb{R}^d$ is computed as same as Eq. (8). Then, the probability for operators corresponding to $t$-th predicted column is computed by softmax classifier as follows:

$$
P_{\text{op}}^{(t)} = \text{softmax}(W_o a_{\text{op}}^{(t)} + b_o) \quad (11)
$$

where $W_o \in \mathbb{R}^{n_o \times d}$ and $b_o \in \mathbb{R}^{n_o}$ are trainable parameters and $n_o$ is the number of possible operators.

### 3.4 From Clause Generation

After the predictions of all the other clauses, we use a heuristic to generate the FROM clause. We first collect all the columns appear in the predicted SQL and then, we JOIN tables that include these predicted columns.

### 3.5 Recursion for Nested Queries

To predict the presence of sub-query, we train another module that has the same architecture of the operator prediction module. Instead of predicting corresponding operators for each column, it predicts whether each column is compared to a variable (ex. WHERE age > 3) or to a sub-query (ex. WHERE age > (SELECT avg(age) ..)). In the latter case, we add the temporal [SUB_QUERY] token to the corresponding location in the SQL output. Additionally, if the sketch generation module predicts one of INTERSECT/UNION/EXCEPT operators, we add [SUB_QUERY] token after the operator.

To predict a sub-query, our model takes the temporal generated SQL with [SUB_QUERY] token as an input in addition to a natural language question with separate token [SEP] (ex.
## Table 1: Accuracy of SQL exact matching with different hardness levels on the dev set.

| Method            | Easy  | Medium | Hard   | Extra Hard | All   |
|-------------------|-------|--------|--------|------------|-------|
| SQLNet            | 23.2% | 8.6%   | 9.8%   | 0%         | 10.9% |
| TypeSQL           | 18.8% | 5.5%   | 4.6%   | 2.4%       | 8.0%  |
| SyntaxSQLNet      | 38.4% | 15.0%  | 16.1%  | 3.5%       | 19.0% |
| Ours              | **53.2%** | **27.0%** | **20.1%** | **6.5%** | **28.8%** |
| -rec              | 53.2% | 27.0%  | 14.4%  | 2.9%       | 27.4% |
| -rec - col-att    | 46.4% | 22.0%  | 12.1%  | 4.7%       | 23.4% |
| -rec - col-att -sketch | 33.2% | 18.6%  | 11.5%  | 4.7%       | 18.7% |

## Table 2: F1 scores of SQL component matching on the dev set.

| Method            | SELECT | WHERE | GROUP BY | ORDER BY | KEYWORDS |
|-------------------|--------|-------|----------|----------|----------|
| SQLNet            | 46.6%  | 20.6% | 37.6%    | 49.2%    | 62.8%    |
| TypeSQL           | 43.7%  | 14.8% | 16.9%    | 52.1%    | 67.0%    |
| SyntaxSQLNet      | 55.4%  | 22.2% | 51.4%    | 50.6%    | 73.3%    |
| Ours              | **68.7%** | **39.0%** | **63.1%** | **63.5%** | **76.5%** |

### 4 Experiments

#### 4.1 Dataset and Evaluation Metrics

We evaluate our model with Spider (Yu et al., 2018c), a large-scale, complex and cross-domain text-to-SQL dataset. We follow the same database split as Yu et al. (2018c), which ensures a database schema appears in the training set does not appear in the dev/test set. Through this split, we examine how well our model can be generalized to unseen databases. Since the test set is not opened to the public, we use the dev set for the experiments. For the evaluation metrics, we use 1) accuracy of exact SQL matching and 2) F1 score of SQL component matching, proposed by (Yu et al., 2018b). We also follow their query hardness criteria to understand the model performance on the different level of queries.

#### 4.2 Model Configuration

We use the same hyperparameters for every module. For the word embedding, we apply deep contextualized word representations (ELMO) from Peters et al. (2018) and allow them to be fine-tuned during the training. For the natural language and column encoders, we use a 1-layer 512-unit bi-directional LSTM. For the decoders in columns and operators prediction modules, we use a 1-layer 1024-unit uni-directional LSTM. For the training, we use Adam optimizer (Kingma and Ba, 2014) with a learning rate 1e-4 and use early stopping with 50 epochs. Additionally, we use dropout (Hinton et al., 2012) with a rate of 0.2 for the regularization.

#### 4.3 Result and Analysis

Table 1 shows the exact SQL matching accuracy of our model and previous state-of-the-art models. We achieve 28.8% on all SQL queries which outperforms the previous best model SyntaxSQLNet (Yu et al., 2018b) by 9.8%. Moreover, our model outperforms previous models on all different hardness of queries.

To examine how each technique contribute to the performance, we conduct an ablation analysis on three aspects: 1) without recursion, 2) without self-attention for database schema encoding, 3) without sketch generation modules that decide the number of decoding steps. Without recursive sub-query generation, the accuracy drops by 5.7% and 3.6% for hard and extra hard queries respectively. This result shows that recursion we use enables the model to predict nested queries. When using final LSTM hidden state as Yu et al. (2018b) instead of using self-attention for schema encoding, the accuracy drops by 4.0% on all queries. Finally, when using only an encoder-decoder architecture without sketch generation for the columns prediction, the accuracy drops by 4.7%.
For the component matching result for each SQL clause, our model outperforms previous approaches for all of the SQL components by a significant margin as shown in Table 2.

5 Conclusion

In this paper, we propose a recursive and SQL clause-wise decoding neural architecture for the complex and cross-domain text-to-SQL generation. We evaluate our model with Spider dataset, and the experimental result shows that our model significantly outperforms previous works for generating not only simple queries but also complex and nested queries.

References

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), volume 1, 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), volume 1, pages 351–360.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), volume 2, pages 732–738.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. arXiv preprint arXiv:1704.08760.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Zhouchan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. arXiv preprint arXiv:1703.03130.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), volume 1, pages 2227–2237.

Patti J Price. 1990. Evaluation of spoken language systems: The atis domain. In Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990.

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.

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

Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What it takes to achieve 100% condition accuracy on wikisql. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1702–1711.

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), volume 2, 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, Qingping 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.
A Sample SQL Predictions

In Table 3, we show some examples of predicted SQL queries from different models. We compare the result of our model with two of previous state-of-the-art models: SyntaxSQLNet (Yu et al., 2018b) and the modified version of SQLNet (Xu et al., 2017) by Yu et al. (2018c) to support complex SQL queries.
Table 3: Sample SQL predictions by our model and previous state-of-the-art models on the dev split. NL denotes the natural language question and Truth denotes the corresponding ground truth SQL query. Ours, Syntax, and SQLNet denote the SQL predictions from our model, SyntaxSQLNet (Yu et al., 2018b), and modified SQLNet (Xu et al., 2017) by Yu et al. (2018c) respectively.