Achieving 90% accuracy in WikiSQL

Wonseok Hwang
wonseok.hwang@navercorp.com

Jinyeong Yim
jinyeong.yim@navercorp.com

Seunghyun Park
seung.park@navercorp.com

Minjoon Seo
minjoon.seo@navercorp.com

Clova AI Research, NAVER Corp., Seongnam, Korea
January 9, 2019

Abstract

In this technical report, we introduce a new state-of-the-art model for natural language to SQL translation task, SQLova. Our model consists of three main modules: (1) BERT-based table- and context-aware word representations [1], (2) a sequence-to-SQL generation module leveraging recent work [2, 3], and (3) execution guided decoding [4] for real-time syntactic and semantic correction. SQLova is evaluated on WikiSQL and achieves 83.6% logical-form accuracy and 89.6% execution accuracy, outperforming previous best models by +8.2% and +2.5% respectively. We plan to open-source our code in near future.

1 Introduction

Semantic parsing is the task of translating natural language utterances to (often machine-executable) formal meaning representations. By helping non-experts to interact with ever-increasing databases, the task has many important potential applications in real life such as question answering and navigation control via speech-based smart devices.

Recently, deep neural networks have achieved new state-of-the-art results in numerous NLP fields: question answering, machine translation, sentiment analysis, et cetera. However, the development of deep learning models in semantic parsing has been limited by the lack of large-scale datasets due to the expensive cost of annotating logical forms. WikiSQL [2] is currently one of the largest hand-annotated semantic parsing datasets consisting of 80,654 pairs of natural language utterances along with their corresponding SQL query annotations on 24,241 tables extracted from Wikipedia.

Here, we present a new model for translating a natural language utterance to SQL query, SQLova, achieving a new state-of-the-art result on WikiSQL. Our model consists of three main modules: (1) BERT-based table- and context-aware word representations [1], (2) a sequence-to-SQL generation module leveraging recent work [2, 3], and (3) execution guided decoding [4] for real-time syntactic and semantic correction. Without the execution guidance, our model achieves 80.7% logical form accuracy on WikiSQL test set, already achieving the state-of-the-art result (+5.3%) on logical form accuracy. With execution guidance, the model shows 83.6% logical form accuracy and 89.6% execution accuracy on WikiSQL test set, outperforming previous best models by 8.2% and 2.5%, respectively. We plan to open-source our code in near future.

2 Related Work

WikiSQL is a large semantic parsing dataset consisting of 80,654 natural language utterances and corresponding SQL annotations on 24,241 tables extracted from Wikipedia [2]. The large size of the dataset

has enabled adopting deep neural techniques for the task and drew much attention in the community recently. The initial baseline proposed by [2] independently generates the two components of the target SQL query, `select`-clause and `where`-clause. SQLNet [3] further simplifies the generation task by introducing sequence-to-set model in which only `where` condition value is generated by sequence-to-sequence model, hence making the model insensitive to the order of the SQL conditions. TypeSQL [5] also employs a sequence-to-set structure but with an additional "type" information of natural language tokens. Coarse2Fine [6] first generates rough intermediate output, and then refines the results by decoding full `where`-clauses. Similarly to SQLova, the final table-aware contextual representation of the question is generated with bi-LSTM with attention. Our model however differs in that many layers of self-attentions [1,7] are employed with a single concatenated input of question and table headers. Pointer-SQL [8] proposes a novel sequence-to-sequence model that uses an attention-based copying mechanism and a value-based loss function. Annotated Seq2seq [9] utilizes a sequence-to-sequence model after automatic annotation of input natural language. MQAN [10] suggests a multitask question answering network that jointly learns multiple natural language processing tasks using various attention mechanisms. Execution guided decoding is suggested in ref. [4], in which non-executable (partial) SQL queries candidates are removed from output candidates during decoding step. IncSQL [11] proposes a sequence-to-action parsing approach that uses incremental slot filling mechanism with feasible actions from a pre-defined inventory.

3 Model

In this section, we describe the details of SQLova, which consists of three parts: (1) table-aware contextualized word representations, (2) sequence-to-SQL generation, and (3) execution-guided decoding [4].

3.1 Table-aware contextualized word representations

Pretrained word representations on a large (unlabeled) language corpus, such as GloVe [12], have shown promising results in WikiSQL [2, 3, 5, 6, 8, 9, 11, 13]. Recently, contextualized word representations [1, 14] drew much attention in the natural language processing community for their superiority over non-contextual methods. Here, we extend BERT [1] for encoding the natural language query together with the headers of the entire table. We use `[SEP]` to separate between the query and the headers. That is, each query input $T_{n,1} \dots T_{n,L}$ (L is the number of query words) is encoded as following:

$$(\texttt{[CLS]}, T_{n,1}, \cdots T_{n,L}, \texttt{[SEP]}, T_{h_1,1}, T_{h_1,2}, \cdots, \texttt{[SEP]}, \cdots, \texttt{[SEP]}, T_{h_{N_h},1}, \cdots, T_{h_{N_h},M_{N_h}}), \quad (1)$$

where $T_{h_j,k}$ is the k-th token of the j-th table header, M_j is the total number of tokens of the j-th table headers, and N_h is the total number of table headers. Each token consists of token embedding, segment embedding, and position embedding. `[CLS]` and `[SEP]` are special tokens for classification and context separation, same as [1]. The output of the encoder is used as the input for our sequence-to-SQL model to be described below.

3.2 Sequence-to-SQL generation

Sequence-to-SQL model is responsible for genearing a target SQL query from the input encoding obtained by our encoder (Section 3.1). In a typical sequence-to-sequence model, the output is not explicitly constrained by any syntax, which is highly suboptimal for formal language generation. Hence, following [3], we adopt syntax-guided sketch, where the generation model consists of six modules, namely `select-column`, `select-aggregation`, `where-number`, `where-column`, `where-operator`, and `where-value`, and we independently train them. We briefly describe what each part is responsible for:

- `select-column` finds `select` column from given natural language utterance,
- `select-aggregation` finds the aggregation operator for the given select column,

- `where-number` predicts the number of `where` conditions in SQL queries,
- `where-column` infers four most probable columns likely to be related with a given natural language utterance,
- `where-operator` finds most probable operators for given `where` column among three possible choices $(>, =, <)$,
- `where-value` finds which tokens of a natural language utterance correspond to condition values for given `where` columns.

Unlike [3], our modules do not share parameters. Instead of using pointer network for inferring the `where` condition values, we train for inferring the start and the end positions of the utterance, following [6]. Furthermore, the inference of start and end tokens in `where-value` module depend on both selected `where-column` and `where-operators` while the inference relies on `where-columns` only in [3].

3.3 Execution-guided decoding

In decoding (SQL query generation) stage, non-executable (partial) SQL queries are excluded from the output candidates following the strategy suggested in [4]. In `select` clause, (`select` column, aggregation operator) pairs are excluded when the string-type columns are paired with numerical aggregation operators such as 'max', 'min', 'sum', or 'avg'. The pair with highest joint probability are selected from remaining pairs. In `where` clause decoding, the executability of each (`where` column, operator, value) pair is tested by checking the answer returned by the partial SQL query `select agg(`$\texttt{col}_s$`) where` $\texttt{col}_w$ `op val`. Here, $\texttt{col}_s$ is the predicted select-column, `agg` is the predicted aggregation operator, $\texttt{col}_w$ is one of the where-column candidates, `op` is where operator, and `val` stands for the `where` condition value. If the query returns an empty answer, it is also excluded from the candidates. The final output of `where` clause is determined by selecting the output maximizing the joint probability estimated from the output of `where-number`, `where-column`, `where-operator`, and `where-value` modules.

4 Experiments

The logical form (LX) and the execution accuracy (X) on dev set (consisting of 8421 queries) and test set (consisting of 15878 queries) of WikiSQL of several models are shown in Table 1. The execution accuracy is measured by evaluating on the answer returned by 'executing' the query on the SQL database. The order of where conditions is ignored in measuring logical form accuracy in our model. The top rows show models without execution guidance, and the bottom rows show models augmented with execution-guided decoding (EG). Our model outperforms previous baselines by a large margin for both non-EG and EG scenarios: 5.3% LF & 2.5 % X for non-EG, and 8.2% LF & 2.5 % X for EG.

Model details Pre-trained BERT models (`BERT-Large-Uncased`[1]) are loaded and fine-tuned with ADAM optimizer with learning rate 10^{-5} whereas the sequence-to-SQL model is trained with 10^{-3} learning rate with $\beta_1 = 0.9, \beta_2 = 0.999$. Batch size is set to 32. To find word vectors, natural language utterance is first tokenized by using Standford CoreNLP [16]. Each token is further tokenized (into sub-word level) by WordPiece tokenizer [1]. The headers of the tables are tokenized by WordPiece tokenizer directly. The PyTorch version of BERT code[2] is used for word embedding and our sequence-to-SQL code is largely influenced by the original SQLNet source code [3]. All computations were done on NAVER Smart Machine Learning (NSML) platform [17, 18]. Our source code will be released soon.

[1] https://github.com/google-research/bert
[2] https://github.com/huggingface/pytorch-pretrained-BERT
[3] https://github.com/xiaojunxu/SQLNet

Table 1: Logical from accuracy (LF) and execution accuracy (X) on dev and test set of WikiSQL.

Model	Dev LF (%)	Dev X (%)	Test LF (%)	Test X (%)
Baseline [2]	23.3	37.0	23.4	35.9
Seq2SQL [2]	49.5	60.8	48.3	59.4
SQLNet [3]	63.2	69.8	61.3	68.0
PT-MAML [15]	63.1	68.3	62.8	68.0
TypeSQL [5]	68.0	74.5	66.7	73.5
Coarse2Fine [6]	72.5	79.0	71.7	78.5
MQAN [10]	76.1	82.0	75.4	81.4
Annotated Seq2seq [9] [1]	72.1	82.1	72.1	82.2
IncSQL [11] [1]	49.9	84.0	49.9	83.7
SQLova (ours)	**81.6** (+**5.5**)	**87.2** (+**3.2**)	**80.7** (+**5.3**)	**86.2** (+**2.5**)
Coarse2Fine-EG [4] [1] [2]	76.0	84.0	75.4	83.8
PointSQL-EG [4] [1], [2]	67.5	78.4	67.9	78.3
IncSQL-EG [11] [1], [2]	51.3	87.2	51.1	87.1
SQLova-EG (ours) [2]	**84.2** (+**8.2**)	**90.2** (+**3.0**)	**83.6** (+**8.2**)	**89.6** (+**2.5**)

5 Conclusion

We introduce SQLova that achieves a new state of the art on WikiSQL. We show that a carefully designed strategy of combining (1) highly contextualized word representation model [1], (2) seq-to-SQL generation [2, 3], and (3) execution-guided decoding [4] can be extremely effective on the task, achieving 90% execution accuracy. We plan to open-source our code in near future. We also plan to release a more detailed paper that includes in-depth model discussion and various analyses and ablations.

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[2] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017.

[3] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," *CoRR*, vol. abs/1711.04436, 2017. [Online]. Available: http://arxiv.org/abs/1711.04436

[4] C. Wang, P. Huang, A. Polozov, M. Brockschmidt, and R. Singh, "Execution-guided neural program decoding," in *ICML workshop on Neural Abstract Machines and Program Induction v2 (NAMPI)*, 2018.

[5] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. R. Radev, "Typesql: Knowledge-based type-aware neural text-to-sql generation," *CoRR*, vol. abs/1804.09769, 2018. [Online]. Available: http://arxiv.org/abs/1804.09769

[6] L. Dong and M. Lapata, "Coarse-to-fine decoding for neural semantic parsing," *CoRR*, vol. abs/1805.04793, 2018. [Online]. Available: http://arxiv.org/abs/1805.04793

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[8] C. Wang, M. Brockschmidt, and R. Singh, "Pointing out SQL queries from text," Tech. Rep. MSR-TR-2017-45, November 2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/pointing-sql-queries-text/

[9] W. Wang, Y. Tian, H. Xiong, H. Wang, and W.-S. Ku, "A transfer-learnable natural language interface for databases," *CoRR*, vol. abs/1809.02649, 2018.

[10] B. McCann, N. S. Keskar, C. Xiong, and R. Socher, "The natural language decathlon: Multitask learning as question answering," *arXiv preprint arXiv:1806.08730*, 2018.

[11] T. Shi, K. Tatwawadi, K. Chakrabarti, Y. Mao, O. Polozov, and W. Chen, "Incsql: Training incremental text-to-sql parsers with non-deterministic oracles," *CoRR*, vol. abs/1809.05054, 2018. [Online]. Available: http://arxiv.org/abs/1809.05054

[12] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[13] H. Xiong and R. Sun, "Transferable natural language interface to structured queries aided by adversarial generation," *CoRR*, vol. abs/1812.01245, 2018. [Online]. Available: http://arxiv.org/abs/1812.01245

[14] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018. [Online]. Available: http://arxiv.org/abs/1802.05365

[15] P. Huang, C. Wang, R. Singh, W. tau Yih, and X. He, "Natural language to structured query generation via meta-learning," in *NAACL*, 2018.

[16] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[17] N. Sung, M. Kim, H. Jo, Y. Yang, J. Kim, L. Lausen, Y. Kim, G. Lee, D. Kwak, J. Ha, and S. Kim, "NSML: A machine learning platform that enables you to focus on your models," *CoRR*, vol. abs/1712.05902, 2017. [Online]. Available: http://arxiv.org/abs/1712.05902

[18] H. Kim, M. Kim, D. Seo, J. Kim, H. Park, S. Park, H. Jo, K. Kim, Y. Yang, Y. Kim, N. Sung, and J. Ha, "NSML: meet the mlaas platform with a real-world case study," *CoRR*, vol. abs/1810.09957, 2018. [Online]. Available: http://arxiv.org/abs/1810.09957