

# Content Enhanced BERT-based Text-to-SQL Generation

Tong Guo<sup>1</sup> Huilin Gao<sup>2</sup>

<sup>1</sup> Never Stop Research

<sup>2</sup> China Electronic Technology Group Corporation Information Science Academy,  
Beijing, China

**Abstract.** We present a simple methods to leverage the table content for the BERT-based model to solve the text-to-SQL problem. Based on the observation that some of the table content match some words in question string and some of the table header also match some words in question string, we encode two addition feature vector for the deep model. Our methods also benefit the model inference in testing time as the tables are almost the same in training and testing time. We test our model on the WikiSQL dataset and outperform the BERT-based baseline by 3.7% in logic form and 3.7% in execution accuracy and achieve state-of-the-art.

**Keywords:** Deep Learning · Semantic Parsing · Database

## 1 Introduction

Semantic parsing is the tasks of translating natural language to logic form. Mapping from natural language to SQL (NL2SQL) is an important semantic parsing task for question answering system. In recent years, deep learning and BERT-based model have shown significant improvement on this task. However, past methods did not encode the table content for the input of deep model. For industry application, the table of training time and the table of testing time are the same. So the table content can be encoded as external knowledge for the deep model.

In order to solve the problem that the table content is not used for model, we propose our effective encoding methods, which could incorporate database designing information into the model. Our key contribution are three folds:

1. We use the match info of all the table cells and question string to mark the question and produce a feature vector which is the same length to the question.
2. We use the match inf of all the table column name and question string to mark the column and produce a feature vector which is the same length to the table header.
3. We design the whole BERT-based model and take the two feature vector above as external inputs. The experiment results outperform the baseline[3]. The code is available.<sup>1</sup>

<sup>1</sup> <https://github.com/guotong1988/NL2SQL-RULE>

| Table:        |     |               |               |                  |                  |
|---------------|-----|---------------|---------------|------------------|------------------|
| Player        | No. | Nationality   | Position      | Years in Toronto | School/Club Team |
| Antonio Lang  | 21  | United States | Guard-Forward | 1999-2000        | Duke             |
| Voshon Lenard | 2   | United States | Guard         | 2002-2003        | Minnesota        |
| Martin Lewis  | 32  | United States | Guard-Forward | 1996-1997        | Butler CC        |
| Brad Lohaus   | 33  | United States | Guard-Forward | 1996-1996        | Iowa             |
| Art Long      | 42  | United States | Guard-Forward | 2002-2003        | Cincinnati       |

|                                     |
|-------------------------------------|
| Question:                           |
| Who is the player that wears No 42? |
| SQL:                                |
| SELECT Player WHERE No. = 42        |
| Answer:                             |
| Art Long                            |

**Fig. 1.** An example of WikiSQL dataset

## 2 Related Work

WikiSQL [1] is a large semantic parsing dataset. It has 80654 natural language and corresponding SQL pairs. The examples of WikiSQL are shown in fig. 1.

BERT[4] is a very deep transformer-based[5] model. It first pre-train on very large corpus using the mask language model loss and the next-sentence loss. And then we could fine-tune BERT on a variety of specific tasks like text classification, text matching and natural language inference and set new state-of-the-art performance on them.

## 3 External Feature Vector Encoding

---

### Algorithm 1 The construction for question mark vector

---

```

vector = [0]*question_length
for cell in table do
    if contains(question,cell) then
        start_index = get_index(question, cell)
        vector[start_index:start_index+len(cell)] = 2
        vector[start_index] = 1
        vector[start_index+len(cell)] = 3
        break
    end if
end for
for one in header do
    if contains(question,one) then
        index = get_index(question, one)
        vector[index] = 4
    end if
end for

```

---

In this section we describe our encoding methods based on the word matching of table content and question string and the word matching of table header and question string. The full algorithms are shown in Algorithm 1 and Algorithm 2. In the Algorithm 1, the value 1 stand for 'START' tag, value 2 stand for 'MIDDLE' tag, value 3 stand for 'END' tag. In the Algorithm 2, we think that the column,

---

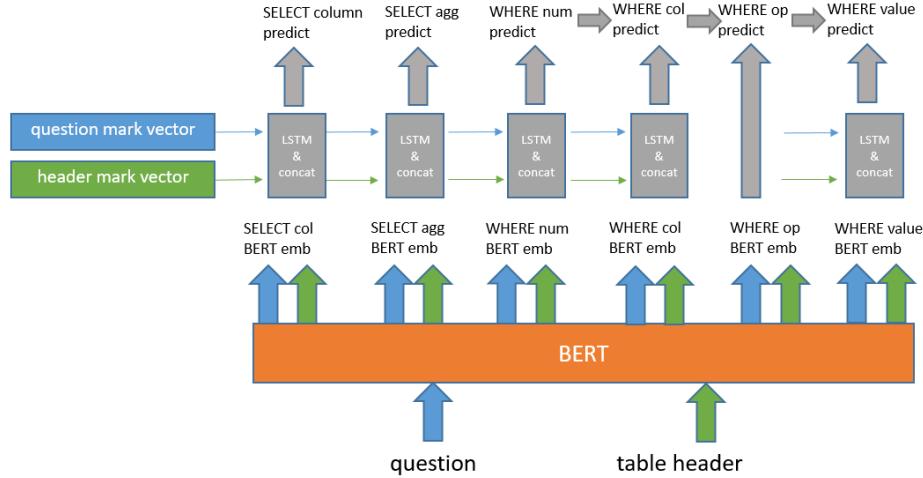
**Algorithm 2** The construction for table header mark vector

---

```
vector = [0]*header_length
index = 0
for one in header do
    if contains(question, one) then
        vector[index] = 1
    end if
    index = index + 1
end for
for cell in table do
    if contains(question,cell) then
        vector[get_column_index(table, cell)] = 2
    end if
end for
```

---

which contains the matched cell, should be marked. The final question mark vector is named  $QV$  and the final table header mark vector is named  $HV$ . For industry application, we could refer to Algorithm 1 and Algorithm 2 to encode external knowledge flexibly.



**Fig. 2.** The deep model

## 4 The Deep Neural Model

Based on the Wikisql dataset, we also use three sub-model to predict the SELECT part, AGG part and WHERE part. The whole model is shown in fig. 2.

We use BERT as the representation layer. The question and table header are concat and then input to BERT, so that the question and table header have the attention interaction information of each other. We denote the BERT output of question and table header as  $Q$  and  $H$

#### 4.1 BERT embedding layer

Given the question tokens  $w_1, w_2, \dots, w_n$  and the table header  $h_1, h_2, \dots, h_n$ , we follow the BERT convention and concat the question tokens and table header for BERT input. The detail encoding is below:

$$[CLS], w_1, w_2, \dots, w_n, [SEP], h_1, [SEP], h_2, [SEP], \dots, h_n, [SEP]$$

The output embeddings of BERT are shared in all the downstream tasks. We think the concatenation input for BERT can produce some kind of 'global' attention for the downstream tasks.

#### 4.2 SELECT column

Our goal is to predict the column name in the table header. The inputs are the question  $Q$  and table header  $H$ . The output are the probability of SELECT column:

$$P(sc|Q, H, QV, HV) \quad (1)$$

where  $QV$  and  $HV$  is the external feature vectors that are described above.

#### 4.3 SELECT agg

Our goal is to predict the agg slot. The inputs are  $Q$  with  $QV$  and the output are the probability of SELECT agg:

$$P(sa|Q, QV) \quad (2)$$

#### 4.4 WHERE number

Our goal is to predict the where number slot. The inputs are  $Q$  and  $H$  with  $QV$  and  $HV$ . The output are the probability of WHERE number:

$$P(wn|Q, H, QV, HV) \quad (3)$$

#### 4.5 WHERE column

Our goal is to predict the where column slot for each condition of WHERE clause. The inputs are  $Q$ ,  $H$  and  $P_{wn}$  with  $QV$  and  $HV$ . The output are the top  $wherenumber$  probability of WHERE column:

$$P(wc|Q, H, P_{wn}, QV, HV) \quad (4)$$

#### 4.6 WHERE op

Our goal is to predict the where column slot for each condition of WHERE clause. The inputs are  $Q$ ,  $H$ ,  $P_{wc}$  and  $P_{wn}$ . The output are the probability of WHERE op slot:

$$P(wo|Q, H, P_{wn}, P_{wc}) \quad (5)$$

#### 4.7 WHERE value

Our goal is to predict the where column slot for each condition of WHERE clause. The inputs are  $Q$ ,  $H$ ,  $P_{wn}$ ,  $P_{wc}$  and  $P_{wo}$  with  $QV$  and  $HV$ . The output are the probability of WHERE value slot:

$$P(wv|Q, H, P_{wn}, P_{wc}, P_{wo}, QV, HV) \quad (6)$$

### 5 Experiments

In this section we describe detail of experiment parameters and show the experiment result.

#### 5.1 Experiment result

In this section, we evaluate our methods versus other approaches on the WikiSQL dataset. See Table 1 and Table 2 for detail. The SQLova[3] result use the BERT-Base-Uncased pretrained model and run on our machine without execution-guided decoding(EG)[6].

**Table 1.** Overall result on the WikiSQL task

| Model            | Logic Form Dev Acc | Execution Dev Acc | Logic Form Test Acc | Execution Test Acc |
|------------------|--------------------|-------------------|---------------------|--------------------|
| SQLova[3]        | 80.6%              | 86.5%             | 80.0%               | 85.5%              |
| Our methods      | 84.3%              | 90.3%             | 83.7%               | 89.2%              |
| Our methods + EG | 85.4%              | 91.1%             | 84.5%               | 90.1%              |

**Table 2.** Break down result on the WikiSQL dataset.

| Model            | SELECT column | SELECT agg | WHERE number | WHERE column | WHERE op | WHERE value |
|------------------|---------------|------------|--------------|--------------|----------|-------------|
| SQLova[3]        | 97.0%         | 90.1%      | 98.5%        | 94.4%        | 97.3%    | 95.5%       |
| Our methods      | 97.4%         | 90.0%      | 99.1%        | 97.9%        | 98.1%    | 97.6%       |
| Our methods + EG | 97.4%         | 90.4%      | 98.9%        | 97.9%        | 97.7%    | 97.9%       |

## 6 Ablation study

The detail results are shown at Table 3. The header vector mainly improve the result of WHERE OP and the question vector mainly improve the result of WHERE VALUE.

**Table 3.** The results of ablation study

| Model                   | SELECT col | SELECT agg | WHERE num | WHERE col | WHERE op | WHERE value |
|-------------------------|------------|------------|-----------|-----------|----------|-------------|
| without all vector      | 97.0%      | 90.1%      | 98.5%     | 94.4%     | 97.3%    | 95.5%       |
| without header vector   | 97.0%      | 90.1%      | 98.5%     | 94.4%     | 97.3%    | 97.3%       |
| without question vector | 97.0%      | 90.1%      | 98.5%     | 97.8%     | 97.3%    | 95.5%       |
| all                     | 97.4%      | 90.0%      | 99.1%     | 97.9%     | 98.1%    | 97.6%       |

## 7 Conclusion

Based on the observation that the table data is almost the same in training time and testing time and to solve the problem that the table content is lack for deep model. We propose a simple encoding methods that can leverage the table content as external feature for the BERT-based deep model, demonstrate its good performance on the WikiSQL task, and achieve state-of-the-art on the datasets.

## References

1. Zhong V, Xiong C, Socher R. Seq2sql: Generating structured queries from natural language using reinforcement learning[J]. arXiv preprint arXiv:1709.00103, 2017.
2. Yu T, Li Z, Zhang Z, et al. Typesql: Knowledge-based type-aware neural text-to-sql generation[J]. arXiv preprint arXiv:1804.09769, 2018.
3. Hwang W, Yim J, Park S, et al. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization[J]. arXiv preprint arXiv:1902.01069, 2019.
4. Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.
5. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.
6. Wang C, Tatwawadi K, Brockschmidt M, et al. Robust text-to-sql generation with execution-guided decoding[J]. arXiv preprint arXiv:1807.03100, 2018.