Formulating Text-to-SQL as a Question-Answering Task

Natasha Singh
singhnat@iu.edu

Abstract

Text-to-SQL interfaces allow users to interact with the databases without actually knowing any query language. They provide a platform where at the frontend, a user can type-in their question in natural language (NLQ) and get the output on the screen and at the backend, the system converts the NLQ to SQL to generate the requested output. Recently, formulating NLP tasks as a Question-Answering task has gained a lot of attention and in my work, I have extended the same approach to the Text-to-SQL task.

1 Introduction

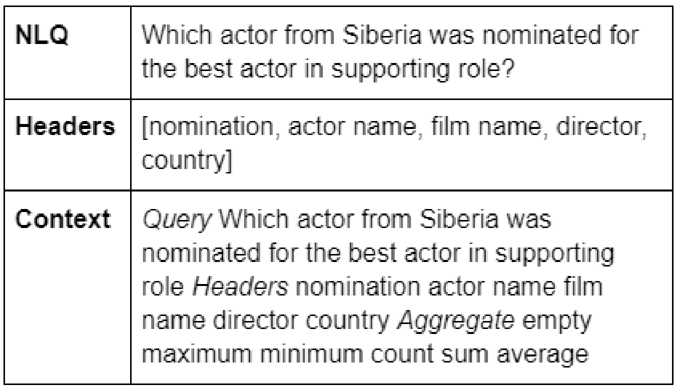
Text-to-SQL generation is a subcategory of Code Generation tasks. Here, our task is to come up with a SQL query given a question in natural language.For example:

NLQ: When did the metrostars[1] have their first rookie of the year[2] winner ?
SQL: SELECT min(season) FROM tbl WHERE team == 'metrostars'

The initial attempts of modeling Natural Languages to generate code / pseudo-codes were all rule based techniques. Dong et al. (2016) formulated the Task of text to SQL as a Seq2Seq model. The Seq2Seq model alone could not perform very well on this task because the model assumes some kind of order in the Sequence, but in reality, the order of filter conditions in the WHERE clause is irrelevant. This was addressed by Zong et al. (2017). They proposed a Seq2SQL model, where they tackled the problem of unordered conditions with Reinforcement Learning. A reward was given when the predicted SQL query generated the same results as the one in ground truth. This model performed quite well, but it overlooked the problem that two semantically different SQL queries can sometimes generate the same results and thus add false rewards for the RL component. So, Xu et al. (2017) avoided the Seq2Seq model with Reinforcement learning (for where clause) for the above stated reasons and came up with SQLNet model which leveraged the inherent structure of SQL. They presented Text-to-SQL as a Slot Filling problem where they used Seq2Set wherever the order of sequence didn't matter. Recently, Yan et al. (2020) used this approach and tried to build SQL query by answering simple, template generated questions like - What is the SELECT column?, What is the aggregate function?, What are the values?, etc.

I have worked on generating simple SQL query using WikiSQL dataset and SQL Template (Fig 1). For each slot in SQL Template, a question is generated and its answer span is extracted from the context (Fig 2, 3). My work is motivated by the work of Yan et al. (2020).

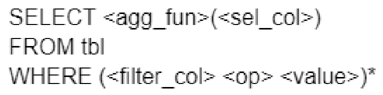
Figure 1: Generating context



NLQ	Which actor from Siberia was nominated for the best actor in supporting role?
Headers	[nomination, actor name, film name, director, country]
Context	*Query* Which actor from Siberia was nominated for the best actor in supporting role *Headers* nomination actor name film name director country *Aggregate* empty maximum minimum count sum average

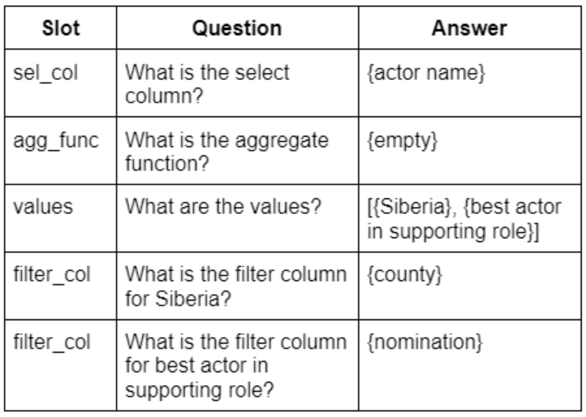
[1]metrostars (now New York red bulls) is a soccer team from New York

[2]rookie of the year award is given to a top-performing athlete in his/her first season within the league



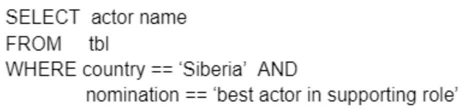
```
SELECT <agg_fun>(<sel_col>)
FROM tbl
WHERE (<filter_col> <op> <value>)*
```

Figure 2: SQL Template



Slot	Question	Answer
sel_col	What is the select column?	{actor name}
agg_func	What is the aggregate function?	{empty}
values	What are the values?	[{Siberia}, {best actor in supporting role}]
filter_col	What is the filter column for Siberia?	{county}
filter_col	What is the filter column for best actor in supporting role?	{nomination}

Figure 3: Template generated questions for each slot and answers extracted for them



```
SELECT  actor name
FROM    tbl
WHERE country == 'Siberia'  AND
          nomination == 'best actor in supporting role'
```

Figure 4: SQL generation

2 Related Work

Yu et al. (2018) put forward the TypeSQL model which was a Knowledge based type-aware neural model. They addressed two major problems in text to SQL in their paper- First, NLQ contains rare entities which are specific to the underlying database and such entities lack accurate embeddings. Second, most of the previous work on this task assumes that user queries contain exact column names and entries. They used type information to understand rare entities (Person, Place, Country, Organisation, Sport) and numbers (Integer, Float, Date, Year) in input.

Huang et al. (2018) attempted modeling this task via Meta Learning. They highlighted that instead of developing a single standard model, it is more beneficial to learn multiple models which specialize in a specific task. In the same paper, they brought out the problem that as the number of tasks increases, each task gets only a handful of training examples for learning a model. To address this problem they suggested adapting to a meta learning framework. They proposed a new method to reduce supervision learning to the few-shot meta learning scenario by introducing a problem based relevance function which helps to group examples to form pseudo tasks.

Wang et al. (2018) proposed a Transfer Learnable Natural Language Interface for Database (NLIDB) protocol to convert NLQ to SQL for any database. In their paper, they learnt and accumulated natural language knowledge and domain specific knowledge independently by separating the meta data (data specific component) from the semantic structure of natural language questions. They achieved this by breaking down the framework into three stages - First, converting NLQ (q) to annotated question form (qa). Second, generating annotated SQL (sa) from qa using a Seq2Seq model. Lastly, converting sa to normal SQL (s).

Dong et al. (2019), in their paper "Data-Anonymous Encoding for Text-to-SQL Generation" talks about two problems that Semantic Parsing tackles - lexical and structural. For a NLQ-to-SQL task, mapping tokens in NLQ to SQL column/cells is a lexical problem and mapping intentions of NLQ to SQL operators/aggregate function/where condition is structural problem. They emphasized on reducing the lexical problem before addressing the structural problem in semantic parsing. For this, they proposed a two-stage anonymization model. In the first stage, they differentiated whether or not a token is related to the table by modeling the lexical task as a sequential tagging problem where each token is tagged either as a column name, cell or nothing. In the second stage, they focused on recognizing the exact column/cell that the token is related to.

Recently, a paper by Xu et al. (2021) enumerated the limitations of implementing a Seq2Seq model for text-to-SQL tasks. The paper focused on improving the performance of Seq2Seq with Schema-aware Denoising (SeaD) by introducing two schema aware noise - erosion and shuffle. Erosion randomly re-orders, adds, and drops columns into the current schema. Shuffle randomly reorders the text in the NLQ. While training, Shuffle is carried out during supervision that trains the model to recover the original NLQ with the correct entity order and Erosion is applied to a Seq2Seq model that trains the model to predict the modified SQL from noised NLQ input.

3 Method

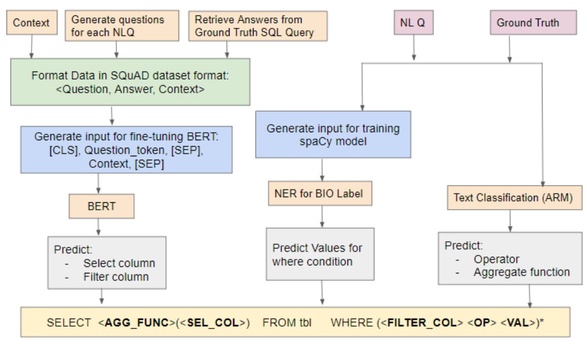

Context
Generate questions for each NLQ
Retrieve Answers from Ground Truth SQL Query
NL Q
Ground Truth
Format Data in SQuAD dataset format: <Question, Answer, Context>
Generate input for fine-tuning BERT: [CLS], Question_token, [SEP], Context, [SEP]
Generate input for training spaCy model
BERT
NER for BIO Label
Text Classification (ARM)
Predict: - Select column - Filter column
Predict Values for where condition
Predict: - Operator - Aggregate function
SELECT <AGG_FUNC>(<SEL_COL>) FROM tbl WHERE (<FILTER_COL> <OP> <VAL>)*

Figure 5: Model Flowchart

3.1 Creating context

Given a natural language query and the table schema (column names), the first step is to generate the context by concatenating NLQ, table headers and SQL aggregate function. They are preceded by the keywords ‘Query’, ‘Headers’ and ‘aggregate’ respectively (Fig 3). For our purpose, we are using five SQL aggregate functions - max, min, count, sum, avg. Some SQL queries do not need an aggregate function, so we have an extra ‘empty’ option.

Some pre-processing like removing the punctuation from the NLQ, lowercasing the text in NLQ and table headers was performed before creating the context to avoid case sensitivity problem while extracting answers from the context.

3.2 Generating Questions using templates

We start by generating three basic questions for each NLQ:

1. What is the select column ?
2. What is the aggregate function ?
3. What are the values ?

After retrieving the answers for the above questions, we generate the question of the following format for each value:

What is the filter column for val ?

The above stated sequence of generating questions is followed only during the testing phase. For the training phrase, these questions are generated simultaneously because we don’t need to wait to extract the answer span from the MRC as the answer is already known.

3.3 Preparing Training dataset in SQuAD format

The Stanford Question Answering Dataset (Rajpurkar et al., 2016) has become the standard data format to train a model for question answering tasks. To represent data into SQuAD format, we transform it into triples of the form (question, answer, context). For our purpose, we form this triplet by using template generated questions for each slot, ground truth/ expected answer for the slot and the context.

3.4 Preparing the input sequence in BERT format

BERT (Delvin et al.., 2019) expects the input sequence to be in [CLS] q1, q2,q3,...,qL, [SEP] c1, c2, c3,. . . cM [SEP] format. It expects the input sequence to be prefixed by the [CLS] token which marks the start of the sequence, followed by question (Q=q1q2q3...qL) and context (C=c1c2c3. . . cM) . A [SEP] token is inserted at the start and end of the context to separate it from the question tokens.

To get the input sequence in this format, I used the ‘bert-large-uncased’ tokenizer from the transformers library by HuggingFace. The longer contexts were truncated to 512 characters and the shorter contexts were extended to 512 characters by adding a [PAD] character. Originally, BERT was trained on WordPiece tokenizer which is a subword tokenization technique that splits the words like playing into play and ##ing. After tokenizing, some of the words in the question/context will be broken down because of this.

3.5 Machine Reading Comprehension (MRC)

For all template generated question (Q), a context (C) is constructed in such a way the answer for each question (predicted slot) is represented by a textual span Cstart-Cend. This text span is extracted from the context using a BERT based MRC. Here, we feed the tokenized input sequence to the BERT and get a contextualized representation matrix (H) as output. The likelihood of each token position being the start and end of the answer span, is calculated using two trainable parameters Vstart and Vend respectively.

$$P_{start}(i) = \text{softmax}(H_i V_{start})$$
$$P_{end}(i) = \text{softmax}(H_i V_{end})$$

The Cstart and Cend are then finally computed by taking the argmax of Pstart and Pend respectively.

3.6 BERT Head for MRC

Yan et al. (2020) trained their MRC model by training BERT on an intermediate task and then finally on the WikiSQL dataset(Phang et al., 2018).

In my work, I adopted the Transfer Learning technique (Perkins, et al., 1992) to simplify the implementation and improve the performance of BERT on WikiSQL. I imported a BERT model pre-trained on Question-Answering task using SQuAD dataset and fine-tuned it on WikiSQL dataset by freezing the first 439 layers and allowing weight updation for the rest of the layers. For training, `token_ids`, `attention_mask`, `start_positions` and `end_positions` were given as input. During Test phase, this fine-tuned model is loaded and is used to extract single answer span (Cstart-Cend) for SELECT column and filter column slots from context by taking `token_ids`, `attention_mask` as input.

3.7 Named Entity Recognition

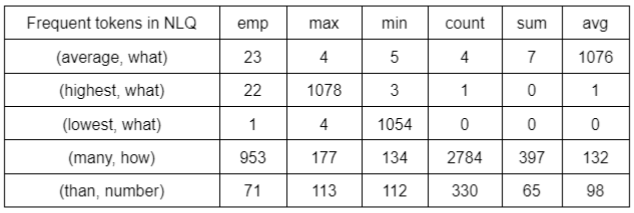
The above described Machine Reading Comprehension method works well for cases where we need to extract only one pair of start and end position from the context, but this model is not good enough for instances where we need to predict multiple values in the query. To overcome this shortcoming, Yan et al. (2020) stacked a CRF layer(Lafferty et al., 2001) on top of BERT to extract multiple text-span.

In my work, I have implemented Named Entity Recognition for extracting multiple named-entities from the text. For this, Training data was prepared in the format: (text, {"entities": [(start, end, label), (start, end, label)]}). I created a blank english model using spaCy library and trained it to recognize entities using BIO tags. During Test phase, this trained model is used to output the named-entities (values of WHERE condition) along with label B/ I to indicate beginning/ continuation of an entity name.

3.8 Text Classification for Aggregate Function and Operator

WikiSQL is a small dataset designed to model Text-to-SQL tasks. On top of this, it has around 10% error in aggregate function [3] annotation (Hwang et al.,2019). These two factors combined explains the poor performance of Aggregate function prediction using MRC. Limited examples in the dataset have imposed similar challenges predicting the operators in WHERE conditions.

Yan et al. (2020) have enhanced aggregate prediction based on a simple rule of the form - "change x to y". In my work, I have implemented Text Classification using Association Rule Mining (Rahman et al., 2010) for predicting the aggregate function slot and operator slot. During the training phase, association rules to classify text are generated by mapping frequently co-occurring tokens with the class that has the highest frequency. Each NLQ in the test phase is then classified into one of ['empty', 'maximum', 'minimum', 'count', 'sum', 'average'] classes based on the association rules derived during training.



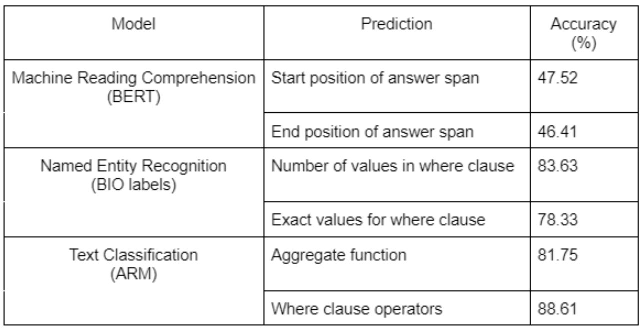
Frequent tokens in NLQ	emp	max	min	count	sum	avg
(average, what)	23	4	5	4	7	1076
(highest, what)	22	1078	3	1	0	1
(lowest, what)	1	4	1054	0	0	0
(many, how)	953	177	134	2784	397	132
(than, number)	71	113	112	330	65	98

Figure 6: Frequency of tokens appearing in emp, max, min, count, sum and average class respectively.

4 Results

For generating final SQL query, column names (select column, filter column) were predicted using BERT based MRC model, values for where clause were extracted from Named Entity Recognition model and aggregate function and operators for where clause were retrieved using the association rules generated while training the Text Classification model. Due to limited time and resources, BERT and NER models were trained for 1 epoch (12hrs) and 12 epochs (6hrs) respectively on google colab. Below is the summary of accuracy of prediction of these models:

Figure 7: Accuracy of models



Model	Prediction	Accuracy (%)
Machine Reading Comprehension (BERT)	Start position of answer span	47.52
	End position of answer span	46.41
Named Entity Recognition (BIO labels)	Number of values in where clause	83.63
	Exact values for where clause	78.33
Text Classification (ARM)	Aggregate function	81.75
	Where clause operators	88.61

[3] An aggregate function performs a calculation on a set of values, and returns a single value

5 Conclusion

This paper formulated Text-to-SQL as a question answering task and used the extracted answers for slot-filling the SQL Template. For this purpose, the slots were predicted by three models specialized in predicting a particular slot (summarized in Fig 7). As future work, I plan on training the BERT based MRC by allowing weight updation of deeper layers and for more epochs to improve accuracy.

6 References

Li Dong, Mirella Lapata. 2016. Language to Logical Form with Neural Attention. ArXiv, abs/1601.01280

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Seq2sql: Generating structured queries from natural language using reinforcement learning. ArXiv, abs/1709.00103.

Xiaojun Xu, Chang Liu, and Dawn Xiaodong Song. 2018. SQLNet: Generating structured queries from natural language without reinforcement learning. ArXiv, abs/1711.04436.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. ArXiv, abs/1804.09769

Zhen Dong, Shizhao Sun, Hongzhi Liu, Jian-Guang Lou, Dongmei Zhang. 2019. Data-Anonymous Encoding for Text-to-SQL Generation. ACL Anthology, D19-1543

Kuan Xuan, Yongbo Wang, Yongliang Wang, Zujie Wen, Yang Dong. 2021. SeaD: End-to-end Text-to-SQL Generation with Schema-aware Denoising. ArXiv, abs/2105.079

Zeyu Yan, Jianqiang Ma, Yang Zhang, Jianping Shen. 2020. SQL Generation via Machine Reading Comprehension. ACL Anthology, 2020.coling-main.31

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, Xiaodong He. 2018. Natural Language to Structured Query Generation via Meta-Learning. ArXiv, abs/1803.02400

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, Ming Zhou. 2018. Semantic Parsing with Syntax- and Table-Aware SQL Generation. ACL Anthology, P18-1034

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. ArXiv, abs/1902.01069

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. ArXiv, abs/1810.04805

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete reasoning. ArXiv, abs/1908.05514

Nitish Shirish Keskar, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Unifying question answering, text classification, and regression via span extraction. ArXiv, abs/1904.09286

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. ArXiv, abs/1412.6980

John D. Lafferty, Andrew McCallum, and Fernando C Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of ICML.

Xiaoya Li, Fan Yin, Zijun Sun, Xiayu Li, Arianna Yuan, Duo Chai, Mingxin Zhou, and Jiwei Li. 2019. Entity-relation extraction as multi-turn question answering. ArXiv, abs/1905.05529

Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. 2020. A unified MRC framework for named entity recognition. ArXiv, abs/1910.11476

Jason Phang, Thibault Fevry, and Samuel R. Bowman. 2018. Sentence encoders on stilts: Supplementary training ´ on intermediate labeled-data tasks. ArXiv, abs/1811.01088.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. ArXiv, abs/1606.05250

D.N. Perkins and G. Salomon. 1992. Transfer of Learning. International Encyclopedia of Education, Second Edition Oxford, England: Pergamon Press

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, Qing He. 2019. A Comprehensive Survey on Transfer Learning. ArXiv, abs/1911.02685

Chowdhury Mofizur Rahman, Ferdous Ahmed Sohel, Parvez Naushad, S. M. Kamruzzaman. 2010.Text Classification using the Concept of Association Rule of Data Mining. ArXiv, abs/1009.4582