Generating Structured Queries from Natural Language

Anand Mohan
moghan.anand@gmail.com

Sourabh V Balgi
sourabhbalgi@gmail.com

Abstract

Relational databases store a significant amount of the world's data which require structured queries like SQL to access it. Synthesizing SQL queries from natural language has been researched over for decades. The problem falls under the category of Natural Language Interfaces (NLI), a research area at the intersection of natural language and human-computer interactions. With the advancements in Deep Learning, it is attracting considerable interests recently. Even though there are variety of papers published in this topic, the accuracies of even the current state of the art solutions are considerably low. We intend to tackle this problem and try to improve the performance on WikiSQL dataset.

1 Introduction

2 Literature Review

Zhong et al. (2017) - They proposed a sequence-to-sequence model - Seq2SQL, a deep neural network for translating natural language questions to corresponding SQL queries. It leverage the structure of SQL to prune the output space of generated queries. Moreover, it uses policy-based reinforcement learning (RL) to generate the conditions of the query, which are unsuitable for optimization using cross entropy loss due to their unordered nature. They have also released the dataset WikiSQL, which we intend to use in our research. They were able to improve the execution accuracy to 59.4% and logical form accuracy to 48.3%.

Xu et al. (2017) - They proposed a sequence-to-set model - SQLNet, which solves the 'order-matters' problem of SQL serialization in sequence-to-sequence model. In addition column attention mechanism to synthesize the query based on sketch was used. They did not use Reinforcement Learning (RL) to reward the decoder when it generates any of the equivalent serializations. Instead, they employed a sketch-based approach where the sketch contains a dependency graph so that one prediction can be done by taking into consideration only the previous predictions that it depends on. They were able to improve the execution accuracy to 68.0% on the WikiSQL dataset.

3 Datasets

We plan to use the WikiSQL, a corpus of 80,654 hand-annotated instances of natural language questions, SQL queries, and SQL tables extracted from 24,241 HTML tables from Wikipedia. (Zhong et al., 2017)

4 Model

4.1 Baseline model

Baseline models for the WikiSQL dataset includes mainly 2 benchmark models.

- *Seq2SQL* (Zhong et al., 2017) : This model is based using reward function for in-the-loop query execution over the database to learn a policy to optimally generate query, which is otherwise difficult to handle with the cross entropy loss because of the unordered parts in the natural language query. Here, the deep reinforcement learning model tries to leverage the inherent structure of the SQL. By applying policy based reinforcement learning with a query execution agent proving the continuous feedback of the rewards for the SQL queries generated, Seq2SQL was shown to outperform state-of-the-art semantic parser with significant improvement in the accuracy metric used for evaluation

- *SQLNET* (Xu et al., 2017) : This baseline model, in contrast to Seq2SQL model in (Zhong et al., 2017), The implementation tries to solve the problem by avoiding the sequence-to-sequence based modelling where the order of the query matters. SQLNET employs a sketch-based approach where the sketch contains a dependency graph so that one prediction is done by taking into account only the previous predictions that it depends on. SQLNET also involves a sequence-to-set model with column-attention mechanism for generating the SQL queries. SQLNET was shown to outperform the Seq2SQL by 9% to 13% .

Since the SQLNET is better among both the benchmark models, SQLNET was used as the baseline model for our implementation.

4.2 Proposed model

The model proposed includes analyzing each components of the SQL query with different LSTM models for generation of the query. The SQL sketch syntax can be represented as *i.e.*
SELECT $AGG $COLUMN
WHERE $COLUMN $OP $ VALUE
(AND $COLUMN $OP $ VALUE)* .
Here, as observed, there are 3 Major components {Aggregate : AGG, SELECT : COLUMN and WHERE : CONDITIONS}. Each of the components are modeled separately using LSTM models.

4.2.1 Aggregate Predictor

This model involves the prediction of the 6 aggregate values {**'NONE', 'MAX', 'MIN', 'COUNT', 'SUM', 'AVG'**}. This involves encoding the tokens in the natural language query using the pre-trained 300-dimension GLOVE embeddings. The imput word embeddings were left-padded to maximum sequence length of the natural language query in the training data. The padded inputs were then fed to LSTM based classifier model to obtain the predicted aggregate value.

4.2.2 Select Predictor

This model involves the prediction of the select columns from 0 to 42 values {**'NONE', 'MAX', 'MIN', 'COUNT', 'SUM', 'AVG'**}. This involves encoding the tokens in the natural language query using the pre-trained 300-dimension GLOVE embeddings. The imput word embeddings were leftpadded to maximum sequence length of the natural language query in the training data. The padded inputs were then fed to LSTM based classifier model to obtain the predicted aggregate value.

5 Experimental Results and Analysis

All the experiments are done on the original WikiSQL dataset by (Zhong et al., 2017). For implementing model to predict the *WHERE* clause, we made some additional data from the existing data. We implemented 3 separate TensorFlow based models for predicts each part of the sql query.

5.1 Embeddings

All the three models train different sets of embedding vectors which are initialized with GloVe(**?**) embeddings. (We used 6 Billion Words, 300 dimensional GloVe embeddings). All tokens not present in GloVe are taken as an *UNK* token and it is initialized with zero vector of 300 dimension and trained. To represent beginning and end of natural language queries, we have used *BEG* and *END* tokens which again are trained.

5.2 AGGREGATE Model

This model employs 2-layered LSTM cells, each of size 128 and the output of which is given to a fully-connected layer of size 128×6(six aggregate operators) and the argmax of the softmax probabilities will give the predicted output. We used a learning rate of 0.01 with decay 0.90. The model was trained in batch sizes of 128 for 20 epochs.

5.3 SELECT Model

The data available is skewed and contained outliers. We removed some samples having randomly occurring columns above index 21 with counts 1. This helped in reducing the prediction space dimension to 22. This model employs 2-layered LSTM cells, each of size 256. Another LSTM cell of 256 is used to encode the column embeddings for column attention. The output after column attention is given to a fully-connected layer of size 256×22(twenty-two columns) and the argmax of the softmax probabilities will give the predicted output. We used a learning rate of 0.001 with decay 0.95. The model was trained in batch sizes of 128 for 30 epochs.

5.4 WHERE Model

This is the most tricky part of sql to predict. For each token in the natural language query, we assign a column and an operator separately.0 is used to represent that the token is not present the *WHERE* clause. We used 2-layered LSTM of size 256 to encode the natural language query. The output of the LSTM layer is connected to 2 separate fully-connected layers of sizes 256×43 (columns) and 256×4(Operators). The softmax of the output is calculated and argmax is taken to get the predicted output. The loss used is average of column loss and operator loss. We used a learning rate of 0.002 with decay 0.90. The model was trained in batch sizes of 128 for 30 epochs.

5.5 Results

The accuracies we obtained are:

AGGREGATE Accuracy	91.3%
SELECT Col Accuracy (w/o CA)	61.2%
SELECT Col Accuracy (CA)	89.4%
WHERE Col Accuracy (w/o CA)	53.7%
WHERE Op Accuracy	63.6%

5.6 Analysis

We tried on with different environments and variants of the proposed model and the following are observations and inferences.

6 Conclusion

7 Future Work

References

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

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning.