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

Semantic parsing is a long-standing open question and has many applications. In particular, it has lots of potential in parsing natural language descriptions into SQL queries. So this has been among the recent trend in NLP tasks attracting much interest from both academia and industry. Since the world's data is mainly represented by relational databases, it has become very important to generate well structured SQL queries from Natural Language Queries. These kinds of tasks can be viewed under Neural Machine Translation using Deep learning models. In particular, in case of SQL queries, we refer to this problem as the natural-language-to-SQL problem (NL2SQL). The de facto standard approach to solve this problem is to treat both the natural language description and SQL query as sequences and train a sequence-to-sequence model or its variants which can be used as the parser. One issue of such an approach is that different SQL queries may be equivalent to each other due to commutativity and associativity. For example, consider the following two queries:

SELECT result **WHERE** score=1-0 AND goal=16
SELECT result **WHERE** goal=16 AND score=1-0

The order of the two constraints in the WHERE clause does not affect the execution results of the query, but syntactically, these two are considered as different queries. It is well-known that the order of these constraints affects the performance of a sequence-to-sequence-style model (Vinyals et al., 2016), and it is typically hard to find the best ordering.

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)

Table: CFLDraft

Pick #	CFL Team	Player	Position	College
27	Hamilton Tiger-Cats	Connor Healy	DB	Wilfrid Laurier
28	Calgary Stampeders	Anthony Forgone	OL	York
29	Ottawa Renegades	L.P. Ladouceur	DT	California
30	Toronto Argonauts	Frank Hoffman	DL	York
...	...	...	...	...

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM
CFLDraft WHERE College = "York"
```

Result:

2

Figure 1: Sample from WikiSQL dataset

4 Model

4.1 Baseline model

Baseline models for the WikiSQL dataset includes mainly 2 benchmark models *Seq2SQL and SQLNET*.

4.1.1 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

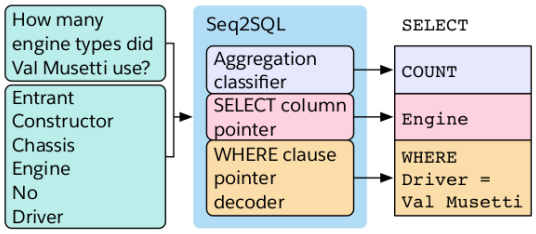

Figure 2: Seq2SQL Model

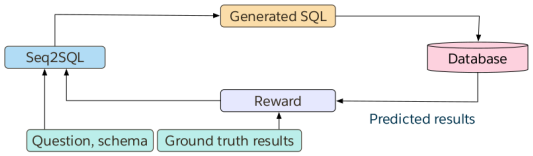

Figure 3: Seq2SQL Architecture

4.1.2 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% .

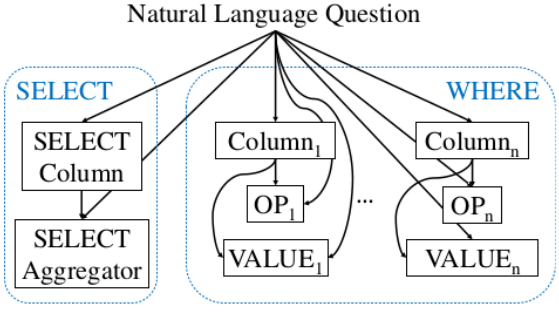

Figure 4: SQLNET Dependency Sketch

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**}.

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	2002-03	Minnesota
Martin Lewis	32, 44	United States	Guard-Forward	1996-97	Butler CC (KS)
Brad Lohaus	33	United States	Forward-Center	1996	Iowa
Art Long	42	United States	Forward-Center	2002-03	Cincinnati

Question:

Who is the player that wears number 42?

SQL:

SELECT player
WHERE no. = 42

Result:

Art Long

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 (Jeffrey Pennington and Manning, 2014) embeddings. The input 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. This involves encoding the tokens in the natural language query using the pre-trained 300-dimension GloVe (Jeffrey Pennington and Manning, 2014) embeddings. The input 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. 2 different approaches were implemented one without column attention and other with column attention.

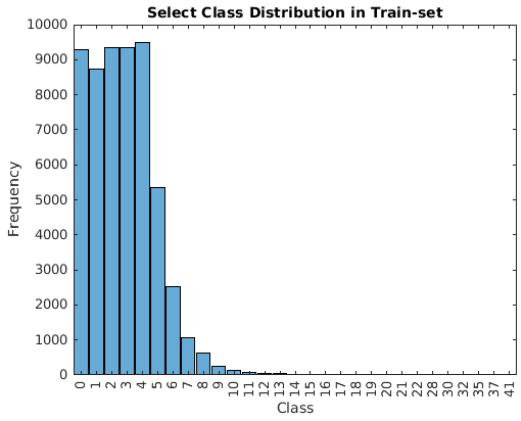

Figure 5: SELECT Column Distribution in Train Set before outlier removal.

Some data samples in the training set with only 1 one occurrence of particular select columns were dropped as this less frequent select column will not be modeled well by the LSTM model resulting in lower validation and test accuracy.

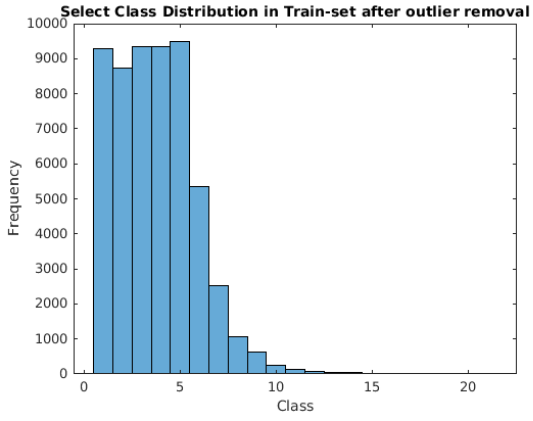

Figure 6: SELECT Column Distribution in Train Set after outlier removal.

4.2.3 WHERE Predictor

This model involves the prediction of the separate entities in the condition $COLUMN $OP

$VALUE. We propose a novel idea of token based COLUMN and OPERATOR prediction using 2 separate models for column and operator. The columns have entries from 0 to 42. There are 4 different operators used {**'NONE', 'EQ', 'LT', 'GT'**}. The tokens in the natural language query are encoded using the pre-trained 300-dimension GloVe (Jeffrey Pennington and Manning, 2014) embeddings. The input 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.

4.2.4 Final SQL query generation

For the final SQL query generation, we combine the outputs of all the 3 models by simple rule based addition. Since SQL queries have structured syntax, simple rule based addition is sufficient to generate robust SQL queries. The SQL queries can be created in the following manner.
SELECT $AGG $SELECT
WHERE $COLUMN $OP $ VALUE
(AND $COLUMN $OP $ 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 (Jeffrey Pennington and Manning, 2014) 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 $\times$ 6 (six aggregate operators) and the 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 with early stopping.

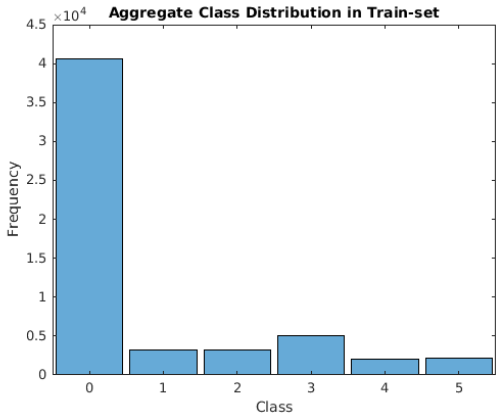

Figure 7: Aggregate Distribution in Train Set

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 $\times$ 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 with early stopping.

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 $\times$ 43 (columns) and 256 $\times$ 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 with

early stopping.

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.

- Left padding of the natural language query is better than right padding it since then all the important data will be to the end and thus the LSTM will get the proper context.
- Performance was better with trainable embedding initialized with GloVe (Jeffrey Pennington and Manning, 2014).
- Removing the outliers in the *SELECT* column, made the data less skewed and improved the accuracy.
- Column Attention improves the accuracy of *SELECT* column with a large margin.
- *WHERE* clause of the SQL query is the most difficult part to predict. It contains mostly the sub-string of the original natural language query. Improving the method of selecting the sub-string will improve the accuracy of the *WHERE* clause.

6 Conclusion

In this paper, we worked on generating SQL queries from natural language. We were able to improve the *AGGREGATE* accuracy by around 1.2 points and we proposed a novel idea to generate the *WHERE* clause which can possibly be improved using Column Attention which we hope to explore in the future. We have evaluated our model on different environments and variants of the proposed model like trainable and non-trainable embeddings, various hyper-parameters, using same embedding and LSTM models for all the three parts and so on.

Also, WikiSQL even though is a large dataset and one of best available dataset, we have faced many discrepancies in the process. A dataset with better database structure and database entries can result in better accuracies. We also suggest adding the additional processed data to the WikiSQL dataset which can be used for more robust prediction of *WHERE* clause.

7 Future Works

One of directions to work in the future is to add a CNN layer before the LSTM layer (CLDNN)(Tara N. Sainath, 2015) for feature extraction which generally gives an improvement of 4-5% in accuracy.

Implementing column attention for the proposed *WHERE* model is another line of future work which can possibly improve the accuracy with a large margin as in the case of *SELECT* model.

References

Richard Socher Jeffrey Pennington and Christoper D. Manning. 2014. Glove: Global vectors for word representation.

Oriol Vinyals Andrew Senior Hasim Sak Tara N. Sainath. 2015. Convolutional, long short-term memory, fully connected deep neural networks.

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.