NL2SQL

Producing SQL Queries from Natural Language

Guillaume Franzoni Darnois - i6256477
Frederic Abraham - i6262598
Yannick Ruppenthal - i6242515

Group 4
Advanced Natural Language Processing

Department of Data Science and Knowledge Engineering
Maastricht University
Netherlands

Abstract

Vast amounts of data are stored in relational databases that are utilized through specific syntax driven queries. In order to make the stored data more easily available to a uneducated user, an interface for natural language is required. Different real-world scenarios can be thought of when addressing this specific task, from a simple chat bot able to collect the requested data from a user, to the more complicated task of performing a search on the semantic web. Our goal is then to inspect the process of creating such interface and the research question becomes the following:

What challenges arise in the development of a model capable of performing the translation from natural language to SQL queries?

To pursue it articulated the project in three main phases. In the first one we collect information about the task and more specifically, about the data sets that are generally used to solve it. Then, in the second one, we look for previous works, select one and attempt to implement it. Finally, we run experiments on the developed architecture, and we evaluate the results.

1 Introduction

The projects main goal is the replication of state-of-the-art techniques in the field of Natural Language Processing. Specifically the task chosen for this project is the NL2SQL one, which consists in the translation of natural language into SQL queries.

This application is particularly suitable for interfacing non-educated people with relational databases, which nowadays contain a vast amount of information. Having a tool capable of transforming natural language into syntax driven queries would mean making many daily processes faster. To put things in perspective, we provide an example:

A financial consultant needs to obtain and observe a company's data to create a financial plan but since they are a consultant, the company's database structure is unknown to them. On top of that they don't have any knowledge of SQL and they wouldn't be able to construct a well-formed query. With a NL2SQL tool it's sufficient to enter the query using natural language and have fast and flawless insights inside the company's data and trends.

The example described above is just one of the many possible applications.

2 Related Work

The problem of translating natural language text into SQL queries has been researched already in depth during the years. Originally known as Natural Language Interfaces to Databases, its development started in the 1980s with attempts to create intermediate logical representations [1]. Those were then improved by using rule-based methods right at the beginning of the 21st century [2]. Nowadays more sophisticated method that make use of Deep Learning have been developed [3, 4].

Along with the latest techniques, the need for more data has led to the development of big data sets like WikiSQL [4] and Spider [5]. The techniques developed are dependent on the data set used because of its queries structures and limitations as mentioned in Section 4.

For this very reason, the choice fell on [3] which uses WikiSQL to transform natural language questions into SQL queries that cannot be nested, grouped, ordered or containing joins.

In general it is possible to distinguish 3 main approaches to the task when WikiSQL is being used:

- **Sequence-to-sequence translation**:
  Trying to model the selection and there where as a sequence-to-sequence

translation doen't preforme perticulart well since the order of the different WHERE and SELECT conditions is not important in an SQL-Querry but in the .

- **Sub-task for different components of SQL queries**:
  The idea followed in this approach is to divide the SQL in multiple and smaller components like the number of SELECT and WHERE conditions, the specific columns for the SELECT and WHERE conditions and the various operators used for those conditions. Main defect is the absence of inter dependencies between the different components that are individually solved through independent models.

- **Exploitation of pre-trained models**:
  Base pre-trained language models are used and fine tuned. This approach is able to capture the inter dependencies left out by the sub-task one.

Their technique uses pre-trained language models, but falls in the second category as we will see in detail in Section 3. The pre-trained model is then fine tuned by following an approach that, differently from the previous methods, does not concatenate the column of the tables in input. Instead a column-wise ranking and decoding is initially performed and recombined in output to form the SQL query by means of straightforward rules.

3 Task & Model

This section describes the specific task of Text-to-SQL and our models. First we show how we preprocess the data into column-question pairs to feed into the pre-trained Bert model. Then we provide a brief description of the individual models used in this task. Finally, we end with the explanation of the rules used to combine them to generate a query given the table structure.

Our main reference is the HydraNet model structure.

3.1 Column-Question Pairs

Provided the columns $c_1, c_2, \ldots, c_n$ with the given question q we create pairs of input sentences for each column named *column-question pairs* [CQP]. Each pair has the following form where type, table name and column name is concatenated and separated with a space.

$$\texttt{[CLS]}\quad \texttt{type}_{c_i}\quad \texttt{table name}\quad \texttt{column name}_{c_i}\quad \texttt{[SEP]}\quad \texttt{Question}\quad \texttt{[SEP]}$$

Using BERTS tokenizer, the special tokens are added, the words are split into subtokens and we receive the embedded id's, attention mask and input type ids for the pretrained model. We require the attention masks as we need to pad the pairs of one question to the same length. The token type ids are used to mark which part of the embedding is for the column and which for the question. One query creates a matrix of CQP.

By combining each column information with the provided question, contrary to the approach of having every column in one input sentence we are able to train more targeted column related task.

3.2 Sub Modules

The approach we are following can be divided into multiple sub tasks. We use six different models each train on a different specific task.

1. The selection model takes every CQP of a request and uses BERT's pooler output of the $[CLS]$ token as input of a linear layer combining them into the probability of being selected as column. Expressed is the probability for the columns as $P(c \mid q) = \texttt{log\_softmax}(w \cdot h_{\texttt{[CLS]}})$ and trained using a negative log likelihood loss.

2. Similarly the model for selecting the most probable where column calculates the probability as $P(c_i \mid q) = \texttt{sigmoid}(w \cdot h_{\texttt{[CLS]}})$ but instead of selecting only the column with the highest value we select based on the number of where columns the top n columns. As more then one column could be correct we use the Binary Cross Entropy loss as reconstruction error.

3. The models for selecting the aggregation (None, MAX, MIN, COUNT, SUM, AVG) and the where condition(=, >, <, OP) can be described as sequence classification task. As input they receive the CQP which are previously selected as either in the select or the where columns. The classification probability is calculated as $P(a \mid c_i, q) = \texttt{log\_softmax}(w \cdot h_{\texttt{[CLS]}})$ using again a negative log likelihood loss for training.

4. The amount of where conditions can be selected by either setting a threshold for the calculated where probability $P(c_i \mid q)$ or by using a classification task proposed by [3]. We use the second approach which calculates the probability of $P(n_w \mid q) = \sum_{c_i} P(n_w \mid c_i, q) \cdot P(c_i \mid q)$. Therefore, we utilize two linear layers. One that calculates the probability of $P(c_i \mid q)$ and another which classifies for each QCP the probability $P(n \mid c_i, q)$ where n is the number of columns with a maximum

of 7 columns. The outputs of the two linear layer are combined using a matrix multiplication and piped through a `log_softmax` layer, trained using a negative loss likelihood loss.

5. The last task is the selection of which part of the question is required as parameter in the where clause. This can be described as a question answer model using sequence extraction. The model produces two outputs, one for the start index and one for the end index. We therefore have two linear layers after the BERT layer with an output size of the specified token length followed by a log_softmax. As a formula $P(x \mid con, c, q) = \texttt{log\_softmax}(w^x \cdot h_{\texttt{[CLS]}})$ where $x \in \{\texttt{start\_idx}, \texttt{end\_idx}\}$, *con* the where condition and c the specified column. Therefore instead of using the original CQP of the selected where column we create a new pair for each column and selected where condition using a more question like approach. Given for example a specified condition "=" then we formulate the input tokens as:

   ```
   [CLS]  column_name  "equal to"  [SEP]  Question  [SEP]
   ```

   For training, we use a cross entropy loss for both indices and take the average for the back propagation step.

3.3 Full Structure

The process of combining the models into the task of creating the SQL-Query is described in figure 1. In the final implementation we pass one shared BERT base model to each task respectively in order to minimise processing time. Therefore, we turn the processes of fine tuning the BERT model off as the individual task would prohibit the correct training.

4 Data

In this section the Data sets used are discussed. In order to do this we briefly present different possible data sets commonly used in the NL2SQL field, as already reported in Kim et al. [6]:

- **WikiSQL**: Developed by Zhong, et al. [4], is by far the biggest database used for this task, it consists of 80.654 pairs of question-queries over 24241 tables extracted from Wikipedia. The queries included in this data set, though, are very simple and they don't allow for grouping, joins, ordering or nested queries. This limits the possibility of a model

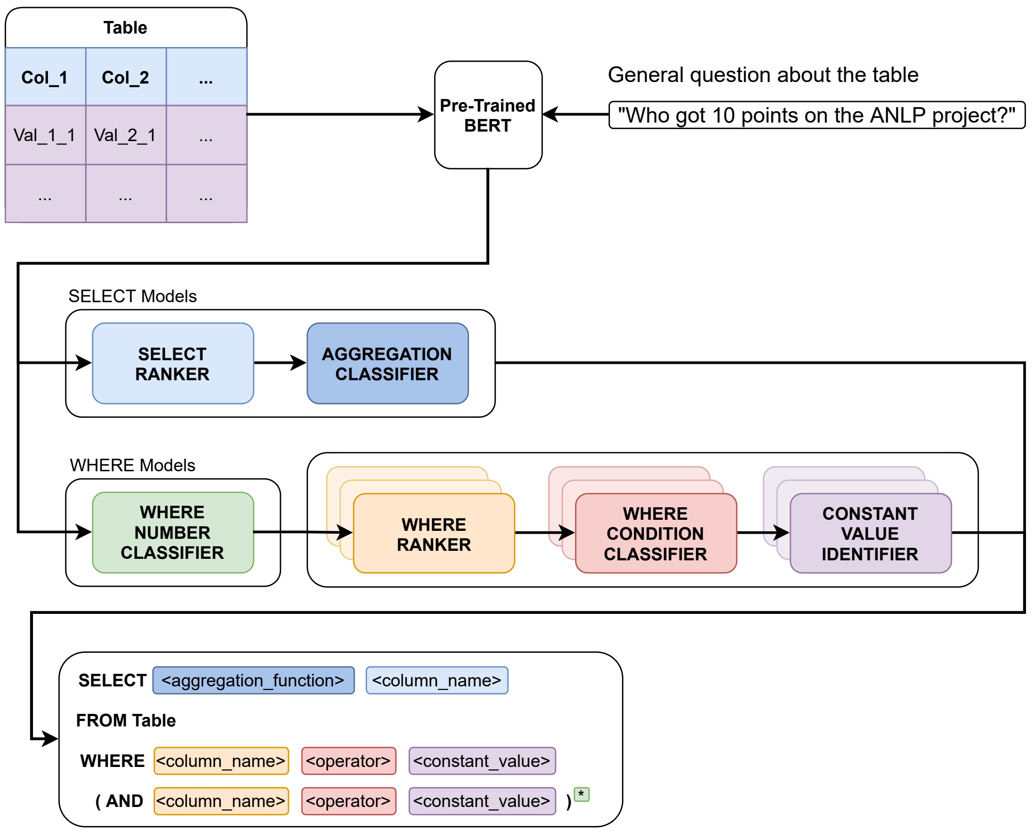
Table
Col_1
Col_2
...
Val_1_1
Val_2_1
Pre-Trained BERT
General question about the table
"Who got 10 points on the ANLP project?"
SELECT Models
SELECT RANKER
AGGREGATION CLASSIFIER
WHERE Models
WHERE NUMBER CLASSIFIER
WHERE RANKER
WHERE CONDITION CLASSIFIER
CONSTANT VALUE IDENTIFIER
SELECT <aggregation_function> <column_name>
FROM Table
WHERE <column_name> <operator> <constant_value>
(AND <column_name> <operator> <constant_value>)*

Figure 1: Combined model to create a query based on

built on top of it but at the same time reduces the complexity of the task.

- **ATIS/GeoQuery**: ATIS and GeoQuery are two single domain data sets. Differently from WikiSQL though, they are more complete, in fact ATIS is missing only grouping and ordering queries while GeoQuery contains every type of query.

- **MAS**: As the previous two data sets, this one is single domain and more complete than WikiSQL, missing only ordering queries. However, the size of it is pretty small, it consists of 196 pairs questions-queries over 17 tables.

- **Spider** Tu et al. [5]: Developed in 2018, proposes itself as a better alternative to both single domain data sets and over simplified ones. With 10,181 questions with 5,693 complex SQL queries spanning over

138 different domains it tries to provide a more complete and sophisticated data set.

As previously mentioned, the goal of the paper is not to implement a new and highly innovative method, but to replicate state of the art technologies to investigate the challenges that arise with their development. In addition to that it is important to notice that because of the nature of these data sets, the techniques used are usually highly dependent on the data used. The final decision resulted on a methodology relying on WikiSQL since plenty of work has been done with it already.

The structure of a query inside WikiSQL is the following:

$$\begin{array}{l}
\text{SELECT } \langle aggregation\_function \rangle \ \langle column\_name \rangle \\
\text{FROM T} \\
[\text{ WHERE } \langle column\_name \rangle \langle operator \rangle \langle constant\_value \rangle \\
\quad (\text{AND } \langle column\_name \rangle \langle operator \rangle \langle constant\_value \rangle)^{*}]
\end{array}$$

It is clear that, as mentioned before, the queries cannot contain grouping, ordering, joins or nested queries. The only operators allowed are Aggregation operators, both for the selection and the condition, and multiple conditions at the same time. In addition it is allowed to have no WHERE conditions at all.

Every Query is then paired with the questions to which it provides the answer. Everything is finally packed into a json structure which is the object used to obtain the data and parse it into a data set that can be loaded with PyTorch. Down below we show how an example looks like in the data set:

```
{
    "phase": 1,
    "table_id": "1-10015132-14",
    "question": "Who played in the Toronto Raptors from 1995-96?",
    "sql": {
        "sel": 0,
        "conds": [[4, 0, "1995-96"]],
        "agg": 0
    }
}
```

Figure 2: Example of question-query pair in WikiSQL

5 Experiments & Discussion

Here we discuss our attempts to train the model to receive good results.

First we tried to initialize every model with its own BERT base model. Therefore, we also tried to fine-tune every BERT base model but even a single pass over the data would have taken 16 hours on a gpu. As previously mentioned as a step to improve training performance we added one shared base model with the weights frozen.

We also tried to utilize different pretrained model that are trained on classification task and question answering task instead of the original BERT base model which is trained on the masking task. The question answering model seemed to achieve a higher accuracy than our own model as we have formulated the CQP as a more question like structure to utilize the pretrained model more effectively. Employing the pretrained model would mean that we need an additional BERT model so we used our implementation with the shared BERT base model.

5.1 Results

We trained on the first phase of the train data set over multiple epochs and for the evaluation As accuracy we took the correct prediction of 500 request examples of unseen data of the first phase. As a base model we used BERT cased model instead of the BERT large uncased that was used by the reference.

Model	S-COL	S-AGG	W-NUM	W-COL	W-OP	W-VAL
SQLova	96.8	90.6	98.5	94.3	97.3	95.4
X-SQL	97.2	91.1	**98.6**	**95.4**	97.6	**96.6**
HydraNet	**97.6**	**91.4**	98.4	95.3	97.4	96.1
Ours:	49.4	63.4	91.4	22.0	**99.7**	57.2

As you can see, we achieve great accuracy in the where operator task but we couldn't get a grasp on the reason that made it reach high values. Probably our evaluation set is to small or the fact that we limited us on the first phase of the data isn't correct. Also, you can see our where ranker didn't achieve any good results probably due to the fact that we used an binary cross entropy loss incorrectly even so the loss over the training data went down.

6 Conclusion

Our goal for this project was to realize a basic implementation of how to translate natural language to an SQL query. To achieve that we took inspiration from the HydraNet Paper. Our results are significantly worse, which was expected considering the scope of our project. This project helped us to get a profound understanding of different tasks in the field of natural language processing. There are multiple aspects which could be improved, like using bigger BERT models, more training, improve our models and add Execution guided decoding.

References

[1] D. H. Warren and F. C. Pereira, "An efficient easily adaptable system for interpreting natural language queries," *American Journal of Computational Linguistics*, vol. 8, no. 3-4, pp. 110–122, 1982.

[2] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates, "Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability," in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, (Geneva, Switzerland), pp. 141–147, COLING, aug 23–aug 27 2004.

[3] Q. Lyu, K. Chakrabarti, S. Hathi, S. Kundu, J. Zhang, and Z. Chen, "Hybrid ranking network for text-to-sql," 2020.

[4] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," 2017.

[5] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," 2019.

[6] H. Kim, B.-H. So, W.-S. Han, and H. Lee, "Natural language to sql: Where are we today?," *Proceedings of the VLDB Endowment*, vol. 13, no. 10, pp. 1737–1750, 2020.

A Training Accuracy and Loss

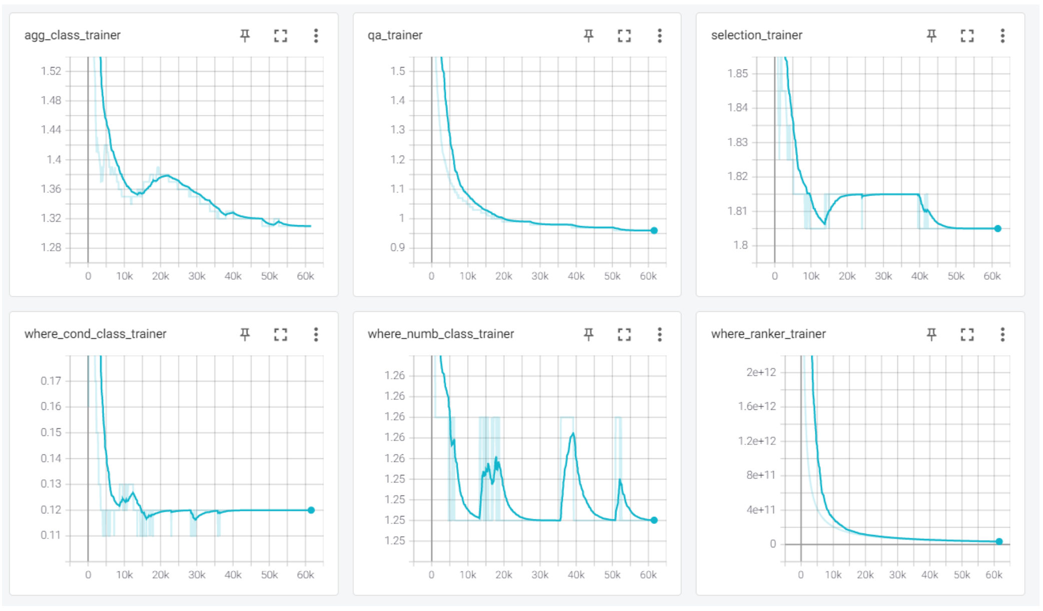

Figure 3: Loss of our model

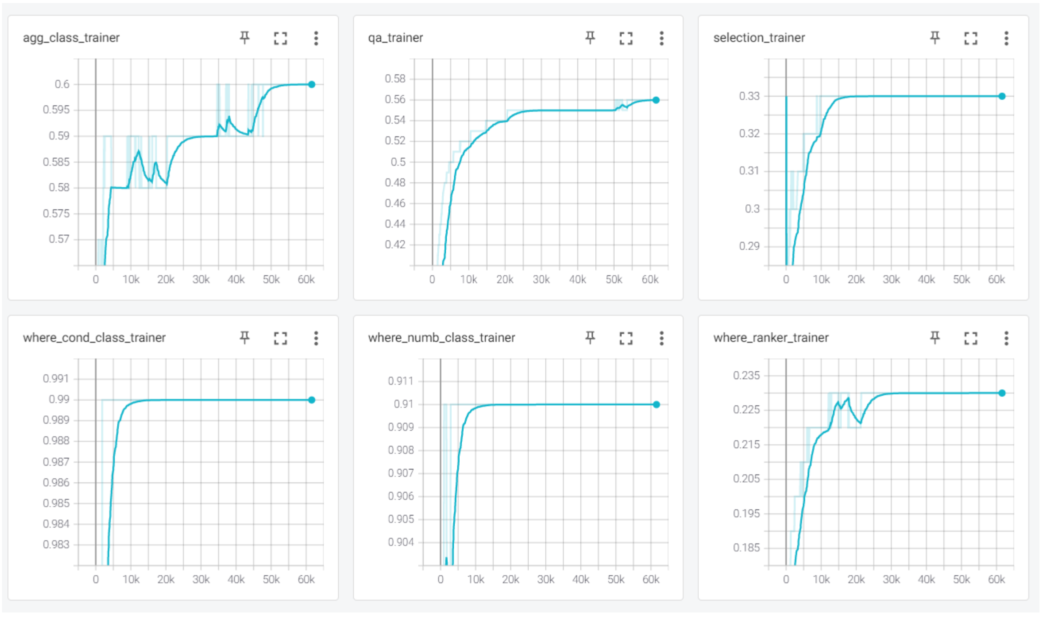

Figure 4: Accuracy of our model

B Example Outputs

1. Columns: ['State/territory', 'Text/background colour', 'Format', 'Current slogan', 'Current series', 'Notes']
   Question: Tell me what the notes are for South Australia

   Query:
   SELECT Notes
   FROM "table_1000181_1"
   WHERE State/territory = "South Australia"

2. Columns: ['Aircraft', 'Description', 'Max Gross Weight', 'Total disk area', 'Max disk Loading']
   question: What if the description of a ch-47d chinook?

   Query:
   SELECT Aircraft
   FROM "Disk loading"
   WHERE Aircraft = "ch - 47d chinook"

3. Columns: ['Player', 'No.', 'Nationality', 'Position', 'Years in Toronto', 'School/Club Team']
   Question: Which school was in Toronto in 2001-02?

   Query:
   SELECT School/Club Team
   FROM "Toronto Raptors all-time roster"
   WHERE Player = "Toronto in 2001 - 02"

4. Columns: ['Player', 'No.', 'Nationality', 'Position', 'Years in Toronto', 'School/Club Team']
   question: which player is from georgia

   Query:
   SELECT Player
   FROM "Toronto Raptors all-time roster"
   WHERE Player = "georgia [SEP] [PAD]"