Natural Language to Structured Query

Dissertation Submitted In Partial Fulfillment Of The Requirements For The Degree Of

Master of Technology
in
Computer Science

by

Soumya Sirkhel
[Roll No: CS1829]

Under the Guidance of

Utpal Garain
Professor, Computer Vision and Pattern Recognition Unit(CVPRU)

Indian Statistical Institute
Kolkata-700108, India

CERTIFICATE

This is to certify that the dissertation entitled **"Natural Language to Structured Query"** submitted by **Soumya Sirkhel** to Indian Statistical Institute, Kolkata, in partial fulfilment for the award of the degree of **Master of Technology in Computer Science** is a *bona fide* record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Utpal Garain
Professor,
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata-700108, India.

Acknowledgment

It was a great privilege and learning experience, to work with Dr. Utpal Garain. He had been a constant source of support, starting from my first year. I want to sincerely thank all the research scholars in NLP Lab . Thanks to my parents and all the friends who had been there with me for the past two years.

Contents

Abstract

Intelligent interface, to enhance efficient interactions between user and databases, is the need of many commercial applications. Oftentimes, users are not familiar with how to frame a structured query as they may not be aware of structure of the database and it is also not expected that the users are required to learn SQL or other query languages to access the database. Hence to simplify task of accessing the database, text-to-SQL models attempt to translate a user's natural language question to corresponding SQL query. Converting natural language to SQL, the model needs to have the ability to create an accurate mapping between the natural language keywords to SQL keywords along with their corresponding tables and columns. Recently, lots of generative text-to-SQL models have been developed. Some of them are using greedy search in their decoder. Hence we choose one of such model[1] and implemented beam search on that. Apart from this we tried explore a discriminative approach for text-to-SQL generation task. A discriminative re-ranker has been proposed on the top of a generative text-to-SQL model for improvement of the accuracy by extracting the best SQL query from a set of beam search predicted candidates. We proposed a schema agnostic discriminative re-ranker built using XLNet fine-tuned classifier for calculating similarity score between natural language and predicted SQL. We used that score to re-rank the beam candidates in a perfect order.

Introduction

1.1 Natural Language to SQL Task

The amount of data produced daily has been increasing exponentially since the start of the new millennia. Most of this data is stored in relational databases. In the past, access to this data has been the interest of mostly large companies, who are able to query the data using structured query languages (SQL). With the growth of mobile phones, more and more personal data is being stored. Thus, more and more people from different backgrounds are trying to query and use their own data. Despite the meteoric rise in the popularity of data science, most people do not have adequate knowledge to write SQL and query their data. Moreover, most people do not have time to learn and understand SQL. Even for SQL experts, writing similar queries, again and again, is a tedious task. Due to this fact, the vast amount of data available today cannot be effectively accessed.

This is where natural language interfaces to databases come in. The goal is to allow you to talk to your data directly using human language! Thus, these interfaces help users of any background easily query and analyze a vast amount of data.

How to Build this Interface? To build this kind of natural language interface, the system has to understand users' questions and convert them to corresponding SQL queries automatically. How can we build such systems? The current best solution is to apply deep learning to train neural networks on a large-scale data of question and SQL pair labels! Compared to rule-based, well-designed systems, these methods are more robust and scalable.

1.2 Dataset

To address the need for a large and high-quality dataset for this task, SPIDER[2] is being introduced by yale university which consists of 200 databases with multiple tables, 10,181 questions, and 5,693 corresponding complex SQL queries. For each database, it covers all the following SQL components: SELECT with multiple columns and aggregations, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, JOIN, INTERSECT, EXCEPT, UNION, NOT IN, OR, AND, EXISTS, LIKE as well as nested queries. In table 1.1 data statistics is shown.

Table 1.1: % of SQL queries that contain a particular SQL component.

WHERE	AGG	GROUP	ORDER	HAVING	SET	JOIN	Nested
55.2	51.7	24.0	21.5	6.7	5.8	42.9	15.7

In this dataset training and the validation data are publicly available. To run and get the result of any model on the test data we have to submit the model to them and they will run the model for us. Because the test data is not publicly available. So in this dissertation we give the result only on the validation dataset. Following the description of the dataset :

- Training data consists of 8659 examples on 146 databases
- Validation data consists of 1034 examples on 20 databases none of which overlaps with the databases in training examples
- Test Data consists of 2147 examples on 40 databases where some of database can overlap with the training database (which we don't know anything apriori)

Why Large, Complex, and Cross-Domain? First, for training a deep learning model, basically, the larger the dataset, the better the performance. Second, training data should cover as many scenarios as possible, including different SQL components and database schema. In this way, a system can learn from them so that the system does not fail in most cases. Finally, why we should care about cross-domain data? Simply, you do not want to relabel data and retrain a new model when you get a new database. This wastes a lot of time!

1.3 Challenges

For building the natural language interface to databases there are three main tasks:

- **Natural language understanding:** The system has to understand users' questions, which could be ambiguous, random and diverse.
- **Database schema representation:** Database can be very complex, with over hundreds of columns, many tables, and foreign keys. The system should understand the representation of the database schema in particular format.
- **Complex SQL decoding/generation:** Once the system understands the user's question and the database's schema to which the user is querying, it has to generate the corresponding SQL query. However, SQL queries can be very complex and include nested queries with multiple conditions.

Few examples from the dataset is described below where we are describing the challenges :

1. - **NL :** What is the hometown of the youngest teacher?
   - **SQL :** SELECT hometown FROM teacher ORDER BY age ASC LIMIT 1

   In the above example model have to identify the word "hometown" as a column the table , "teacher" in "course_teach" database. The database will be given at along with each query. Similarly the ORDER BY keyword and ASC is being inferred from "youngest".

2. - **NL :** Find the age of students who **do not have** a cat pet.
   - **SQL :** SELECT age FROM <u>student</u> WHERE student NOT IN (SELECT ... FROM <u>student</u> JOIN <u>has_pet</u> ... JOIN <u>pets</u> ... WHERE ...)
   - **NL :** What are the names of teams that **do not have** match season record?

- **SQL :** SELECT name FROM team WHERE team id NOT IN (SELECT team FROM match season)

The above two pair of examples from SPIDER showing how similar questions can have different SQL queries, conditioned on the schema. Table names are underlined.

3. - **NL :** Show the names of students who have a grade higher than 5 and have at least 2 friends.

   - **SQL :** SELECT T1.name FROM friend AS T1 JOIN highschooler AS T2 ON T1.student_id = T2.id WHERE T2.grade > 5 GROUP BY T1.student_id HAVING count(*) >= 2

This example is showing the mismatch between the intent expressed in NL and the implementation details in SQL. The column 'student_id' to be grouped by in the SQL query is not mentioned in the question.

Evaluation Metric

2.1 BLEU Score and it's Drawbacks

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU. Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality.

Intelligibility or grammatical correctness are not taken into account in BLEU score. Now, SQL is is structured query and the each keyword in the query is very important along with their positions. Since the structure and grammatical correctness is not captured in the BLEU score, this is not a good evaluation metric in text-to-SQL evaluation task. For example :

- Gold SQL : SELECT country , count(*) FROM singer GROUP BY country
- Predicted SQL : SELECT count (*) , country FROM singer GROUP BY country

BLEU score = 0.58 but the two SQLs are actually same.

2.2 Evaluation metric for text-to-SQL task

In this section we are going to describe the evaluation metric for text-to-SQL task. This include Component Matching[2] and Exact Matching[2]. In addition, the system's accuracy measured as a function of the difficulty of a query. This evaluation metrics do not take value strings into account.

2.2.1 Component Matching

To conduct a detailed analysis of model performance, the average exact match between the prediction and ground truth is measured for each of the following SQL components:

• SELECT • WHERE • GROUP BY • ORDER BY • KEYWORDS (including all SQL keywords without column names and operators)

The prediction and the ground truth has been decomposed into the above components as bags of

Table 2.1: examples of SQL decompositions

SQL	SELECT city
	FROM employee
	WHERE age <30
	GROUP BY city
	HAVING count (*) >1

SELECT	{ NO DISTINCT FLAG , (SELECT , NO AGGREGATE OP , employee . city) }
WHERE	{ (WHERE , employee . age , <, VALUE) }
GROUP BY	{ (GROUP BY , employee . city , HAVING COLUMN = employee . _all_ , count , >=) }
ORDER BY	NONE
KEYWORDS	{where , group by , having }

SQL	SELECT name , capacity
	FROM stadium
	ORDER BY average DESC
	LIMIT 1

SELECT	{NO DISTINCT FLAG , (SELECT , NO AGGREGATE OP , stadium.name), (SELECT , NO AGGREGATE OP , stadium.capacity) }
WHERE	NONE
GROUP BY	NONE
ORDER BY	{ (ORDER BY , stadium. average , NO AGGREGATE OP , DESC , LIMIT = 1) }
KEYWORDS	{ desc, order by, limit }

several subcomponents, and check whether or not these two sets of components match exactly. To evaluate each SELECT component, for example, consider SELECT avg(col1), max(col2), min(col1), we first parse and decompose into a set (avg, min, col1), (max, col2), and see if the gold and predicted sets are the same. Previous work directly compared decoded SQL with gold SQL. Some examples of the decomposition of SQLs are given in the table 2.1 .

However, some SQL components do not have order constraints. In the evaluation, each component is treated as a set so that for example, SELECT avg(col1), min(col1), max(col2) and SELECT avg(col1), max(col2), min(col1) would be treated as the same query. But both the predicted and ground truth should have tables from the same database otherwise the evaluation metric will regard the predicted SQL as completely wrong SQL and treat the predicted SQL as a null SQL. (i.e. every component is filled with NONE) which automatically results no accuracy improvement in each component.

2.2.2 Exact Matching

To measure whether the predicted query as a whole is equivalent to the gold query, first evaluate the SQL clauses as described in the last section. The predicted query is correct only if all of the components are correct. Because set comparison has been done in each clause such that exact matching metric can handle the “ordering issue”.

2.2.3 SQL Hardness Criteria

To better understand the model performance on different queries, SQL queries are divided into 4 levels: easy, medium, hard, extra hard. The difficulty is defined based on the number of SQL components, selections, and conditions, so that queries that contain more SQL keywords (GROUP BY, ORDER BY, INTERSECT, nested subqueries, column selections and aggregators, etc) are considered to be harder. In the following manner the hardness is defined :

- SQL components 1: WHERE, GROUP BY, ORDER BY, LIMIT, JOIN, OR, LIKE, HAVING
- SQL components 2: EXCEPT, UNION, INTERSECT, NESTED
- Others: number of agg > 1, number of select columns > 1, number of where conditions > 1, number of group by clauses > 1, number of group by clauses > 1 (no consider col1-col2 math equations etc.)

Then different hardness levels are determined as follows.

1. Easy: if SQL key words have ZERO or exact ONE from [SQL components 1] and SQL do not satisfy any conditions in [Others] above. AND no word from [SQL components 2].
2. Medium: SQL satisfies no more than two rules in [Others], and does not have more than one word from [SQL components 1], and no word from [SQL components 2]. OR, SQL has exact 2 words from [SQL components 1] and less than 2 rules in [Others], and no word from [SQL components 2]
3. Hard: SQL satisfies more than two rules in [Others], with no more than 2 key words in [SQL components 1] and no word in [SQL components 2]. OR, SQL has 2 ¡ number key words in [SQL components 1] ¡= 3 and satisfies no more than two rules in [Others] but no word in [SQL components 2]. OR, SQL has no more than 1 key word in [SQL components 1] and no rule in [Others], but exact one key word in [SQL components 2].
4. Extra Hard: All others left.

Related Works

Converting natural language to the corresponding SQL is a machine translation task. The whole work on text-to-SQL can be divided into two parts - (1) Simple Rule Based and (2) Deep Learning based.

3.1 Simple Rule Based Works

In rule based system, SQLs are predicted from a fixed set of grammar rules. In the paper of (Li et al.)[3], first the natural language will be parsed, then there is an interactive communicator which will communicate with user telling which part of natural language is mapped to which part of the database. Then user will insert or discard some part of that input manually. After that, natural language parser generates best K parse tree and tell the user about each of them in natural language for taking the user's choice. So user will choose best parsed tree for the natural language. After that chosen parse tree will be mapped to SQL according to the pre-defined grammar.

3.2 Deep Learning Based Works

Deep learning text-to-SQL task can be viewed as a neural machine translation from natural language to SQL using encoder-decoder. In both encoder and decoder part RNN or LSTM or GRU can be used. Encoder encodes the natural language and decoder predicts the corresponding SQL.

In the paper of (Hosu et al.)[4] they used dual encoder structure for predicting the SQL from natural language. First they mask the column names with <column> token, table name with <table> token, any value string with <string> and any numerical value with <num> token. Then they train and predict the structure of the SQL with masked column name, table name and value strings/numbers. After that, they used dual encoder, feeding natural language to one encoder and SQL to the other to finally predict the SQL. Idea for using dual encoder is that they will take the corresponding column name and table from the natural language in the position of the masks.

In SQLNet[5] the basic idea is to employ a sketch, which highly aligns with the SQL grammar. Therefore, SQLNet only needs to fill in the slots (select_column, select_aggregator, where_column1, where_op1, where_value1,...) in the sketch rather than to predict both the output grammar and the content. In this architecture column names in the where clause and select clause will be predicted using attention mechanism with the question and the columns in the databases. This is first of this kind of technique in this field.

In GNN[6] they used Graphical neural network in schema linking along with encoder-decoder model. They used LSTM encoder-deocder along with attention on the input. Instead of predicticting the SQL keywords and column names or table names at decoder they predict pre-defined grammar rules. At each decoding step, a non-terminal is expanded using one of the grammar rules. Rules are either schema-independent and generate nonterminals or SQL keywords, or schema-specific and generate schema items.Then schema-specific items will be predicted by graph neural network.

IRNet[7] is used for tackling the mismatch problem and the lexical problem with intermediate representation and schema linking. Specifically, instead of end-to-end synthesizing a SQL query from a question, IRNet decomposes the synthesis process into three phases. In the first phase, IRNet performs a schema linking over a question and a schema. The goal of the schema linking is to recognize the columns and the tables mentioned in a question, and to assign different types to the columns based on how they are mentioned in the question. Incorporating the schema linking can enhance the representations of question and schema, especially when the out of domain words lack of accurate representations in neural models during testing. Then, IRNet adopts a grammar-based neural model to synthesize a SemQL query, which is an intermediate representation (IR) to bridge NL and SQL. Finally, IRNet deterministically infers a SQL query from the synthesized SemQL query with domain knowledge.

Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions[1] exploits the idea of correlation between sequentially generated queries and generalize the system to different domains. They propose query generation by editing the query in the previous turn. They first encode the previous query as a sequence of tokens, and the decoder computes a switch to change it at the token level. This sequence editing mechanism models token-level changes and is thus robust to error propagation. Furthermore, to capture the user utterance and the complex database schemas in different domains, they use an utterance-table encoder based on BERT to jointly encode the user utterance and column headers with co-attention, and adopt a table-aware decoder to perform SQL generation with attentions over both the user utterance and column headers.

We are choosing this model as our generative architecture. Since SPIDER dataset is not context dependent (i.e. doesn't depend on the previous query), this methods have slight modifications that we will be discussing in the next chapter.

Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions

4.1 Task Formulation

Let X denote a natural language utterance and Y denote the corresponding SQL query. In context independent scenario an interaction, I is considered consisting of n utterance query pair in a sequence $I = [(X_i, Y_i)]_{i=1}^{n}$. At each turn t, the goal is to generate Y_t given the current utterance X_t. Furthermore, in the cross-domain setting, the model is also given the schema of the current database as an input. Let us consider a relational databases with multiple tables, and each table contains multiple column headers: $T = [c_1, c_2, ..., c_l, ..., c_m]$ where, m is the number of column headers (i.e. column name in the database) and each c_l consists of multiple words including its table name and column name.

4.2 modules

Encoder-Decoder architecture with attention mechanisms has been applied as illustrated in Figure 1. The framework consists of the following parts :

1. an utterance-table encoder to explicitly encode the user utterance and table schema,
2. a table-aware decoder taking into account the context of the utterance, the table schema.

4.2.1 Utterance-Table Encoder

An effective encoder captures the meaning of user utterances, the structure of table schema, and the relationship between the two.

Utterance Encoder

Figure 4.1(b) shows the utterance encoder. For the user utterance at each turn, They first use a bi-LSTM to encode utterance tokens. The bi-LSTM hidden state is fed into a dot-product attention layer (Luong et al., 2015) over the column header embeddings. For each utterance token embedding, attention weighted average of the column header embeddings is calculated to obtain the most relevant columns (Dong and Lapata, 2018).Then the bi-LSTM hidden state and the column attention vector is concatenated , and a second layer biLSTM is used to generate the utterance token embedding $\mathbf{h^E}$.

Table Encoder

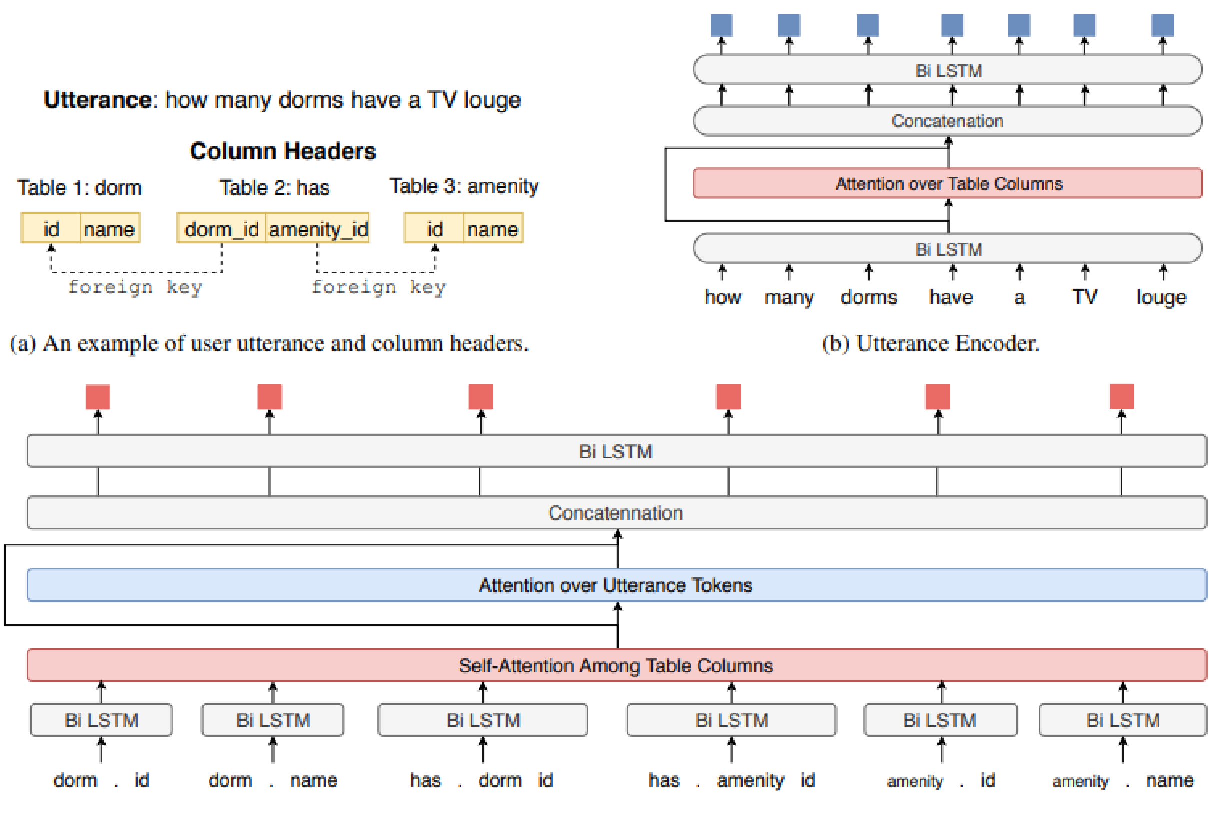
In figure 4.1(c) shows the table encoder. For each column header, its table name and its column name is concatenated separated by a special dot token (i.e., table name . column name). Each column header is processed by a bi-LSTM layer. To better capture the internal structure of the table schemas (e.g., foreign key), a selfattention (Vaswani et al., 2017) have been employed among all column headers. Then an attention layer is used to capture the relationship between the utterance and the table schema. At last the self-attention vector and the utterance attention vector is concatenated, and a second layer bi-LSTM is used to generate the column header embedding $\mathbf{h^C}$.

Utterance-Table BERT Embedding

There are two options as the input to the first layer biLSTM. The first choice is the pretrained word embedding. Second, the contextualized word embedding based on BERT (Devlin et al., 2019) can also be considered. In the second case the embedding is generated as follows:

$$[CLS], X_i, [SEP], c_1, [SEP], ..., c_m, [SEP]$$

This sequence is fed into the pretrained BERT model whose hidden states at the last layer is used as the input embedding.



(a) An example of user utterance and column headers.

(b) Utterance Encoder.

(c) Table Encoder.

Figure 4.1: Utterance-Table encoder

4.2.2 Table-aware Decoder

LSTM decoder with attention is used to generate SQL queries by incorporating the current user utterance, and the table schema. Denote the decoding step as k, decoder's input is concatenation of the embedding of SQL query token q_k and a context vector c_k: $\mathbf{h}^D_{k+1} = LSTM^D([\mathbf{q}_k; \mathbf{c}_k], \mathbf{h}^D_k)$,where $\mathbf{h}^D$ = hidden of $LSTM^D$, $\mathbf{h}^D_0$ is initialised by $\mathbf{h}^E_{t,|X_t|}$. When the query token is a SQL keyword, q_k is a learned embedding; when it is a column header, we use the column header embedding given by the table-utterance encoder as q_k. The context vector c_k is described below.

Context Vector with the Table and User Utterance

The context vector consists of attentions to both the table and the user utterance. First, at each step k, the decoder computes the attention between the decoder hidden state and the column header embedding

$$\begin{aligned} s_l &= \mathbf{h}^D_k \mathbf{W}_{column-att} \mathbf{h}^C_l \\ \alpha^{column} &= softmax(s) \\ \mathbf{c}^{column}_k &= \sum_l \alpha^{column}_l \times \mathbf{h}^C_l \end{aligned} \tag{4.1}$$

where l is the index of column headers and $\mathbf{h}_l^C$ is its embedding. Second, it also computes the attention between the decoder hidden state and the utterance token embeddings:

$$\begin{aligned} s_{i,j} &= \mathbf{h}_k^D \mathbf{W}_{utterance-att} \mathbf{h}_{i,j}^E \\ \alpha^{utterance} &= softmax(s) \\ \mathbf{c}_k^{token} &= \sum_{i,j} \alpha_{i,j}^{utterance} \times \mathbf{h}_{i,j}^E \end{aligned} \tag{4.2}$$

where i is the turn index, j is the token index, and $\mathbf{h}_{i,j}^E$ is the token embedding for the j-th token of i-th utterance. The context vector c_k is a concatenation of the two , $\mathbf{c} = [\mathbf{c}_k^{column}; \mathbf{c}_k^{token}]$

Output Distribution

In the output layer, our decoder chooses to generate a SQL keyword (e.g., SELECT, WHERE, GROUP BY, ORDER BY) or a column header. This is critical for the crossdomain setting where the table schema changes across different examples. To achieve this, separate layers are used to score SQL keywords and column headers, and finally the softmax operation is used to generate the output probability distribution:

$$\begin{aligned} \mathbf{o}_k &= tanh([\mathbf{h}_k^D; \mathbf{c}_k]\mathbf{W}_o) \\ \mathbf{m}^{SQL} &= \mathbf{o}_k \mathbf{W}_{SQL} + b_{SQL} \\ \mathbf{m}^{column} &= \mathbf{o}_k \mathbf{W}_{column} \mathbf{h}^C \\ P(y_k) &= softmax([\mathbf{m}^{sql}; \mathbf{m}^{column}]) \end{aligned} \tag{4.3}$$

Proposed Work

5.1 Shortcomings and Modification on Editing-Based SQL Query Generationfor Cross-Domain Context-DependentQuestions

The model, described in the chapter 4, is using greedy search in it's decoder. Hence, We have implemented Beam Search on it. As a result we are getting an improvement of .02% on exact matching for the validation set. So the proposed exact matching accuracy was 57.6% and we are getting 57.8%.

Table 5.1: Accuracy with different beam size

Beam Size	Exact Matching Accuracy (%)
1	57.8
2	58.9
5	69.1
10	71.9
20	74.4
40	75.9

From table 5.1, we can see that the exact matching accuracy increases along with beam size,which is quite intuitive but this much significant improvement from beam size 2 to beam size 5 is not been expected. This could be due to structured query prediction from a fixed vocabulary size (i.e. SQL keywords and column headers).

5.2 Baseline Model

As a baseline model, we experimented with a SEQ2SEQ model along with attention on SPIDER[2] dataset. We used the open-source neural machine translation toolkit, OpenNMT[8] which implements a standard SEQ2SEQ model with global attention[9]. LSTM cells with two hidden layers and 500 neurons is used in both the encoder and decoder. The word embedding layer has 500 neurons and maximum batch size was 64. We used Stochastic Gradient Descent (SGD) to train the model for 25 epochs with learning rate = 1.0 and learning decay = 0.5.

5.3 Motivation for our Discriminative Model

In most of the previous works, researchers are putting too much of preference in generative model for solving text-to-SQL task. It is easier to choose correct SQL from a set of SQLs candidates rather than predicting the correct one from the given schema and utterance . From 5.1 we can conclude that a decoded SQL with highest log probability can not always be correct. That is why, we are getting an exact matching accuracy improvement from beam size = 1 to beam size = 5. Table 5.2 shows that "**SELECT Citizenship , count (*) FROM singer GROUP BY Citizenship ORDER BY sum (Net_Worth_Millions) DESC limit 1**" ,having the highest log probability among the other beam candidates, is not the correct SQL. But the SQL with second highest log probability is the correct one and matched with the gold SQL.

QUESTION: For each citizenship, what is the maximum net worth?	
Gold SQL: SELECT Citizenship , max(Net_Worth_Millions) FROM singer GROUP BY Citizenship	
Beam Candidates	Log Probability
SELECT Citizenship , count (*) FROM singer GROUP BY Citizenship ORDER BY sum (Net_Worth_Millions) DESC limit 1	-0.11448371566867772
SELECT Citizenship , max (Net_Worth_Millions) FROM singer GROUP BY Citizenship	-0.1365424266343265
SELECT Citizenship , sum (Net_Worth_Millions) FROM singer GROUP BY Citizenship	-0.13699122393618954
SELECT Citizenship , avg (Net_Worth_Millions) FROM singer GROUP BY Citizenship	-0.2125829908972126
SELECT Citizenship , count (*) FROM singer GROUP BY Citizenship ORDER BY sum (Net_Worth_Millions) DESC =	-0.8557227556184401

Table 5.2: Beam candidates with beam size = 5 sorted by their log probabilities.

The above case is happened due to wrong prediction of log probabilities in some cases. Hence we could fix those probabilities by calculating a similarity score between between natural language utterance and their predicted SQLs. Then we will choose the SQL from the beam candidates having maximum similarity score. Now for our experiment throughout this section we are assuming beam size = 5 due to hardware constraints.

5.4 Our Method

5.4.1 Intuition

If we have a generative model which is taking database as an input and predicting all the SQLs from their respective database along with very huge improvement in accuracy using beam size>1 (like the case discussed in table 5.1) , then we could apply a schema agnostic re-ranking model on the top of the generative model to re rank the SQLs perfectly. If the GOLD SQL is present in the beam candidates then it should get a highest score value and other NON-GOLD SQLs will get lower score such that the GOLD SQL present in the beam candidates would end up at the top if we order them by descending score values. That's the main goal of this discriminative model. Since the schema information is already incorporated in the generative model, then the SQLs predicted by that model is already aware of the schema information. Apart from that, in most of the cases schema information is needed to predict the perfect table names and column names of SQLs. So the main structure of an SQL should come from the utterance. That's why, we are preferring a schema agnostic re-ranker for ordering the beam candidates in a perfect manner.

5.4.2 Model Architecture

We build a Discriminative Reranker model as a binary classifier by fine tuning XLNet[10], to predict whether a given candidate query, s is the gold query for given utterance, u and schema information, D. Since we are using the model architecture described in chapter 4 as a generative model, hence we are not taking schema information into account in our model. So, our model is schema agnostic.

Let u be the utterance and S=$\{s_1, s_2, s_3, s_4, s_5\}$ be the set of generative model predicted SQLs using beam search[5.1]. Now we pair up those SQLs with their utterance u, i.e. $\{(u, s_1), (u, s_2), (u, s_3), (u, s_4), (u, s_5)$
We build up our model using the following steps :

1. **Preprocessing :**First, utterance u and SQL query s_i are encoded using XLNet. We use $< SEP >$ to separate utterance and the SQL query. We use SentencePiece [11] to tokenize utterance into utterance tokens $u_1, ..., u_N$ and SQL query into query tokens $s_{i_1}, ..., s_{i_M}$, where N and M are the token counts for utterance and SQL query, respectively. The tokens are combined as follows to form the input token sequence:

   $$< SEP > +u_1 + ... + u_N + < SEP > +s_{i_1} + ... + s_{i_M} + < SEP > + < CLS >$$

   We did pre-padding for making all the utterance-SQL pair of same length.

2. We take the last layer's hidden state for the last token, $H_{<CLS>}$, as the input embedding and passed into a linear layer with *tanh* activation function to form a pooled encoding.

3. The resulting encoding is passed through a linear classification layer and we used BCEloss (binary cross entropy loss)to train the model.

Hence, the mathematical formulation of similarity score between utterance and SQL is given below,

$$\textit{sim-score}(u, s_i) = \textit{log}(\frac{p}{1-p})$$

where, p = probability of being Gold SQL ; $(1 - p)$ = probability of being Non-Gold SQL. Hence the range of similarity socre, *sim-score* is $[-\infty, \infty]$.

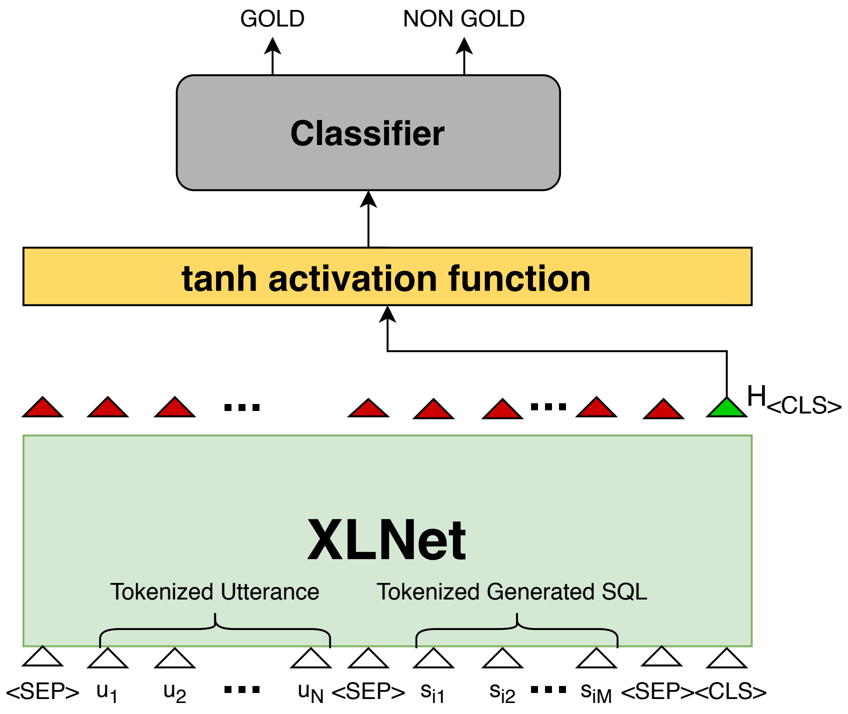


Figure 5.1: Model Architecture

5.4.3 Implementation Details

First we train our modified generative model 5.1 and save the model parameters . Then we generate the 5 SQLs per utterance (i.e. beam size = 5) for both the training set and validation set using beam search and saved model parameters. After that we train our discriminative model by feeding the utterance-SQL pair generated for the training dataset. At the time of inference, we choose those SQLs from the beam candidates having maximum *sim-score* among the other beam candidates.

Our model uses XLNet-base model. We use a pre-trained XLNet model from Hugging face library[12] and fine-tune it at the time of training using following hyper-parameters:

- **Optimizer :** Adam
- **learning rate :** 2e-5
- **learning rate decay :** linear
- **weight decay :** 0.01

Results and Discussions

6.1 Examples

Few examples where our discriminative model is performing better than the generative model[1]:

1. - **Question :** What are each professional's first name and description of the treatment they have performed?
   - **Gold SQL :** SELECT DISTINCT T1.first_name , T3.treatment_type_description FROM professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id JOIN Treatment_types AS T3 ON T2.treatment_type_code = T3.treatment_type_code
   - **Generative Model Output :** SELECT first_name , last_name FROM Professionals
   - **Beam Candidates:**
     - * **SQL :** SELECT T2.first_name , T2.last_name FROM Treatments AS T1 JOIN Professionals AS T2 ON T1.professional_id = T2.professional_id ORDER BY T1.cost_of_treatment
       * sim-score : -3.761164
     - * **SQL :** SELECT T1.first_name , T4.size_description FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id JOIN Dogs AS T3 ON T2.dog_id = T3.dog_id JOIN Sizes AS T4 ON T3.size_code = T4.size_code
       * sim-score : -1.9178727
     - * **SQL :** SELECT first_name , first_name FROM Professionals
       * sim-score : -4.773424
     - * **SQL :** SELECT T1.first_name , T3.treatment_type_description FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id JOIN Treatment_Types AS T3 ON T2.treatment_type_code = T3.treatment_type_code
       * sim-score : 2.054987
     - * **SQL :** SELECT first_name , last_name FROM Professionals
       * sim-score : 1.7634317

In the above examples sim-scores calculated by our discriminative method is performing better than the generative model. In the first example Generative Model is predicting "SELECT first_name , last_name FROM Professionals" which is not the correct SQL. But the Gold SQL is in the beam candidates. According to our motivations the Gold SQL should get the highest score than the others. Here, "SELECT T1.first_name , T3.treatment_type_description FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional_id = T2.professional_id JOIN Treatment_Types AS T3 ON T2.treatment_type_code = T3.treatment_type_code" is getting the highest score which is matching with Gold SQL.

2. - **Question :** List each owner's first name, last name, and the size of his for her dog.
   - **Gold SQL :** SELECT T1.first_name , T1.last_name , T2.size_code FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id
   - **Generative Model Output :** SELECT T1.first_name , T1.last_name , T3.size_description FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id JOIN Sizes AS T3 ON T2.size_code = T3.size_code
   - **Beam Candidates:**
     - * **SQL :** SELECT T1.first_name , T1.last_name , T3.size_description from Owners as T1 join Dogs as T2 on T1.owner_id = T2.owner_id join Sizes as T3 on T2.size_code = T3.size_code where T2.name = 1
       * sim-score : 0.3081195
     - * **SQL :** SELECT first_name , last_name , last_name FROM Owners
       * sim-score : -4.8888183
     - * **SQL :** SELECT T1.first_name , T1.last_name , T2.size_code FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id
       * sim-score : 1.2493361
     - * **SQL :** SELECT T1.first_name , T1.last_name , T3.size_description FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id JOIN Sizes AS T3 ON T2.size_code = T3.size_code
       * sim-score : 0.8786468
     - * **SQL :** SELECT T1.first_name , T1.last_name , T3.size_code FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id JOIN Sizes AS T3 ON T2.size_code = T3.size_code
       * sim-score : 0.6024068

Similarly, in the second example "SELECT T1.first_name , T1.last_name , T3.size_description FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id JOIN Sizes AS T3 ON T2.size_code = T3.size_code" is the output predicted by the generative model which is not correct. But if we apply the our discriminative method on the top of the generative model[1] then the beam candidate, "SELECT T1.first_name , T1.last_name , T2.size_code

FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id" is getting highest sim-score among all other beam candidates and it is matching with the Gold SQL.

6.2 Detail Results of generative and discriminative models

	Easy	Medium	Hard	Extra Hard	Overall
Count	250	440	174	170	1034
Exact Matching	0.768	0.588	0.425	0.265	0.551
Component Matching					
select	0.912	0.825	0.919	0.800	0.858
select(no AGG)	0.940	**0.840**	0.925	0.836	0.877
where	0.836	0.731	0.490	0.476	0.659
where(no OP)	0.868	0.744	0.584	0.595	0.716
group(no Having)	0.785	0.782	**0.861**	0.825	0.803
group	0.725	0.739	***0.833***	0.786	0.765
order	0.786	0.700	0.850	0.744	0.757
and/or	1.000	0.959	**0.953**	0.910	0.960
IUEN	0.000	0.000	0.306	**0.448**	0.320
keywords	0.937	0.913	0.803	0.737	0.862

Table 6.1: Detail Result for our Discriminative model

	Easy	Medium	Hard	Extra Hard	Overall
Count	250	440	174	170	1034
Exact Matching	0.780	0.609	0.466	0.335	0.578
Component Matching					
select	0.920	0.826	0.925	0.813	0.864
select(no AGG)	0.944	0.838	0.931	0.843	0.880
where	0.864	0.779	0.523	0.516	0.700
where(no OP)	0.882	0.779	0.614	0.626	0.743
group(no Having)	0.850	0.831	0.833	0.861	0.841
group	0.850	0.788	0.833	0.819	0.809
order	0.815	0.728	0.891	0.756	0.784
and/or	1.000	0.963	0.942	0.928	0.963
IUEN	0.000	0.000	0.333	0.400	0.325
keywords	0.955	0.927	0.835	0.776	0.885

Table 6.2: Detail Result of the modified model [1]
described in 5.1

We already get 0.2% of improvement in exact matching accuracy by implementing beam search in the model[1] which is already discussed in 5.1. Now in table 6.2 and table 6.1 we are presenting detail results of the underlying generative model [1] modified by us [5.1] and our discriminative model respectively.

Since the accuracy have been computed component wise, if all the components in the predicted

SQL matched with all the components in the Gold SQL then only exact matching will increase by one. After getting an improvement in exact matching accuracy from beam size = 2 to beam size = 5 (table 5.1) we have expected that our Discrminative model should be able to captue the correct SQL among the beam candidates and perform better than the generative model. Now if we compare both the results present in the table 6.1 and table 6.2 , we see that our Discriminative model can not outperform the generative model ,modified by us, in all level of hardness of SQLs. But if we compare the results for two models in component matching, we get the accuracy improvement in "select (no AGG)" in Medium SQLs, "group (no Having)" in Hard SQLs, "IUEN" in Extra Hard SQLs. Along with this we are getting same component matching accuracy in "group" in Hard SQLs.

6.3 Comparison of results and Conclusion

Table 6.2 shows the performance of various methods on the validation set of SPIDER dataset[2].

In our baseline model we didn't use the database schema information for predicting SQLs. That's why it is giving very lowest exact matching accuracy, 0.1%.

Since we are incorporating with database schema in our baseline model, we looked into "Editing-based sql query generation for cross-domain context-dependent questions" [1] which applies greedy search at decoder. So we modified that model and implemented beam search on it and get improvement on exact matching accuracy, from 57.6% to 57.8%.

After that we observed improvement in accuracy in beam size = 5. Hence we build a discriminative model which is giving exact matching accuracy, 55.1%.

Table 6.3: Comparison of accuracy of models of SPIDER dataset

Model Name	Exact Matching (in percent)
IRNet + BERT	61.9
EditSQL + BERT with beam search (our modification on decoder)[5.1]	**57.8**
EditSQL + BERT	57.6
Our Discriminative Model[5.4]	**55.1**
GNN	40.7
SQLNet	10.9
SEQ2SEQ OpenNMT (our baseline model)[5.2]	**0.1**

Future Scope

In this thesis, we looked into the implementation of the related works and choose "**Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions**" model for SPIDER task where we have proposed beam search decoding and a schema agnostic discriminative model. The final result shows that the proposed model gives comparable result with the state-of-art architecture. There are lots of scope for improvement of the proposed model:

- If we order the scores of beam candidates in decreasing order then if the deviation of scores between two consecutive candidates is less than a threshold they will change their order of preference.
- Other than this discriminative structure one can use reinforcement learning to get the scores.
- The best could be using adversarial machine translation with generator as the previously described model 4. Since in this text-to-SQL task no one ever tried adversarial machine translation (such as GAN) before.

Bibliography

[1] Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. Editing-based sql query generation for cross-domain context-dependent questions. *arXiv preprint arXiv:1909.00786*, 2019.

[2] Kai Yang Michihiro Yasunaga Dongxu Wang Zifan Li James Ma Irene Li Qingning Yao Shanelle Roman Zilin Zhang Dragomir Radev Tao Yu, Rui Zhang. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

[3] Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

[4] Ionel Alexandru Hosu, Radu Cristian Alexandru Iacob, Florin Brad, Stefan Ruseti, and Traian Rebedea. Natural language interface for databases using a dual-encoder model. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 514–524, 2018.

[5] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[6] Ben Bogin, Matt Gardner, and Jonathan Berant. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*, 2019.

[7] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[8] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[9] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[10] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[11] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[12] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, pages arXiv–1910, 2019.