CSI4900 Report
Generating Structured Queries from Natural Language

Junbo Tang(8639271);

Yi Zhao(8650881)

Supervisor:Dr.Diana Inkpen

April 2021

[2]

Contents

1 Introduction

1.1 Context

There is plenty of valued data stored in databases; and it is widely used in many industries, such as stock data, medical records, and so on. However, there is a certain difficulty in using these data for most people who are not familiar with Structured Query Language (SQL). Thus, today transferring natural language (NL) to SQL is a very promising and hot research area.

We chose to study the first model "Seq2SQL" and the second newest model "SQLova". We study and compare these two models in principles and structures, and also do some modifications to see the change. The "Seq2SQL" is based on a very classic and famous model "Seq2Seq", which is applied to all general NLP. There are also many improvements and changed model-based it to achieve high accuracy, it could help us to understand the principle and the general process of transfer NL to SQL. The "SQLova" is a model with second high accuracy, we compare these two models to see how to improve the accuracy and how does the model evolve step by step from the beginning. We do not choose the newest technology X-SQL from Microsoft, because the paper does not disclose too many details and there is no open-source code. SQLova gets advantages of BERT and using the concept of Sequence to Set(Section 6.3) and Column Attention(Section 6.4) from SQLNet. Therefore, we also did an in-depth study in SQLNet and get a clear picture of how the researchers share their knowledge and making progress step by step.

1.2 Goals

1. Learn different models like Seq2Seq, Seq2SQL, SQLNet, SQLova and see how does each mechanism improve the accuracy step by step.
2. Understand the general and basic process of NLP model
3. Have an in-depth understanding of "Seq2SQL", "SQLNet", and "SQLova" structures and algorithms.
4. Learn and understand the database "WikiSQL" and how to evaluate the performance of trained models for transfer NL to SQL
5. Make some modifications to the original "Seq2SQL" algorithm to see the changes

1.3 Why is it important

Text-to-SQL is currently a hot area of NLP research, a powerful and efficient Text-to-SQL algorithm can provide tremendous help for human work, and it can also reduce the threshold of getting information from data. There are several machine learning models which can transfer the natural language (English) to SQL, which could help people search data from databases in natural language. The first model "sequence to SQL" (Victor Zhong,2017) put forward by Salesforce Research with accuracy 57.1%, the newest model "X-SQL+EG" (Pengcheng He, 2019) with accuracy 91.8%, we can obviously see that the accuracy improved significantly. There are also ten more models between these two which cannot be enumerated one by one.

2 Wiki SQL

WikiSQL consists of a corpus of over 80 thousand hand-annotated SQL query and natural language question pairs. The models we are going to introduce are all trained with WikiSQL. Figure 1 [8] shows WikiSQL is the largest hand-annotated semantic parsing dataset to date.

Dataset	Size	LF	Schema
WikiSQL	**80654**	**yes**	**24241**
Geoquery	880	yes	8
ATIS	5871	yes*	141
Freebase917	917	yes	81*
Overnight	26098	yes	8
WebQuestions	5810	no	2420
WikiTableQuestions	22033	no	2108

Figure 1: An Comparison between Different Models(Vzhong et al., 2017)

In the Wiki SQL data set, the table name is given. The model can directly get the table name. Therefore, the FROM clause doesn't need to be trained. Each query consists of only one table, therefore, Join operation is not needed in the data set. Aggregation in the SELECT clause include {", 'MAX', 'MIN', 'COUNT', 'SUM', 'AVG'} and the VALUE slot in the WHERE clause is chosen from {=, >, <}

Table:

Player	Country	Points	Winnings ($)
Steve Stricker	United States	9000	1260000
K.J. Choi	South Korea	**5400**	756000
Rory Sabbatini	South Africa	3400	4760000
Mark Calcavecchia	United States	2067	289333
Ernie Els	South Africa	2067	289333

Question: What is the points of South Korea player?
SQL: SELECT Points WHERE Country = South Korea
Answer: 5400

Figure 2: An example of generated query using Wiki SQL(Hwang et al., 2019)

The input of the task is a natural language question and a table, the output is a SQL query or an executed result. Here the natural language question is what are the points of South Korean player, The generated query selects points where the country is equal to South Korea and gets the correct answer which is five thousand and four hundred.

3 Recurrent Neural Network (RNN) and Long Short-term memory (LSTM)

3.1 RNN

The very common model for solving the sequence problem is RNN. From the bellowing graph, we can see that the network runs repeatedly; if it is expanded, it looks like the right part, a sequence structure. At each execution, it takes one character and the previous state as its input adds its information to the state vector. In this case, more and more information will be kept in the state. Obviously, we can find that has a good fitting for the sequence resource, like the text, since both the network and source have the same sequence structure.

The inside structure is very simple, it only contains one activation function (1), a hyperbolic tangent function (tanh) which takes X_t and the previous state vector h_{t-1} as input, which squashes the activation to the range between -1 and 1; and then it calculates the current state vector ht. Where W_t is the weight of each parts.

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{1}$$

3.2 LSTM

However, RNN got a potential problem. If the input is too long, then the very beginning information would be lost since more and more information was added to the state. To solve this kind of problem, Sepp Hochreiter and Jiirgen Schrnidhuber put forward an inspired improvement long short-term memory [7]. It still keeps the sequence structure but redesigns the inside part.

Three different gates were added in each cell to determine how much information was added and was left. The key point for each gate is keeping the important information and forget the non-necessary part.

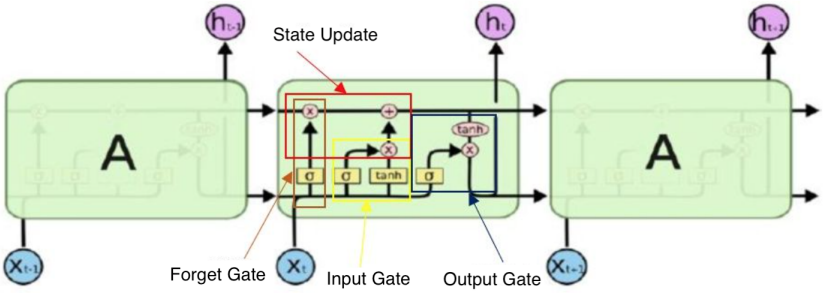

Figure 3: The insider structure of LSTM (Hochreiter et al., 1997)

1. Forget Gate

The forget gate decides how much previous information will be forgotten based on the last output and new input. It uses sigmoid function which will return a number between 0 and 1 as probability, to determine it.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

2. Input Gate
   The input gate determines how much new information will be added to the state vector C. It will keep the part that will play a vital role in the subsequently, e.g., the part which describes the limitation factor of the whole question query. It separates into two steps, the Sigmoid function decides the value which is added firstly, and then the tanh function gives the weight of the values which are passed deciding their level of importance ranging.

$$i_t = \sigma(W_i \cdot ([h_{t-1}, x_t] + b_i) \tag{3}$$

$$\widetilde{C_t} = tanh(W_c \cdot ([h_{t-1}, x_t] + b_C) \tag{4}$$

3. State Update
   The new state will be updated based on the result of these two gates. First, using the old state vector multiplies the result of "forget gate" ft to get the necessary part of the previous state, and then added the new information it×Ct˜ on it to update the state.

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C_t} \tag{5}$$

4. Output Gate
   Finally, it calculates the step output based on the new state. The output is different from the state, most of the time, output only contains parts of the state which will be needed in the current step. It uses the modified by Sigmoid function input to determine how much information will be concluded, and then use it multiplies the state to get the output.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = o_t \times tanh(C_t) \tag{7}$$

4 Sequence-to-Sequence

4.1 Seq2Seq

The first general and common frame for solving the NLP problem is the Seq2Seq network. The basic construction contains two parts, an "Encoder" and a "Decoder" (As shown in Figure 4 [4]). Each part is whether using RNN or LSTM algorithm.

The general idea is to use the dense vector as an intermediate variable to transfer one language to another language. Firstly, the Encoder compressed an input X (X_1,X_2, ... X_n) with N-dimensions to a dense vector 'C' or call it "hidden layer" as a final state, which contains all

necessary information in it. Secondly, the Decoder translates the "hidden layer" to result in sequence step by step., and it will get an M-dimensions Y(Y_1,Y_2, ... Y_n) output. In each step, the decoder finds the most possible outcome based on the previous sequence.

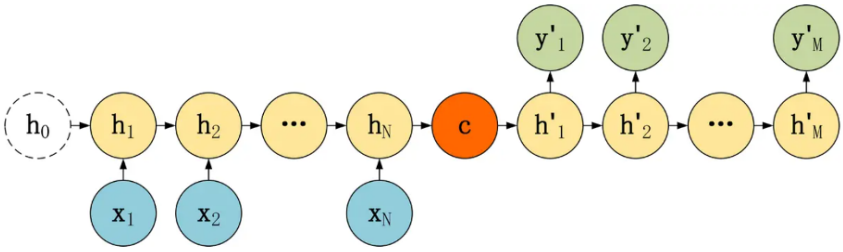

Figure 4: General Seq2Seq Model (Sutskever et al., 2014)

The very important specialty of seq2seq is the length of input and output can be different, it is an N-to-M model, which is very important progress compare with the traditional models, which only have the fixed input and output. Since the code compiles the structure as a recurrent structure, as shown in section 3.1, for the decoder part, when all the input is entered, it will stop reading automatically without assigning any length requirement. For the decoder part, when the <end>tag is returned, the sequence will be stopped. This mechanism lets the network have the ability to handle all length input and could return any length output.

4.2 Attention mechanism

The attention mechanism (Figure 5 [3]) is a technology that allows the model to focus on important information and fully absorb it. It is a technology that can be applied to any sequence model instead of a complete model.

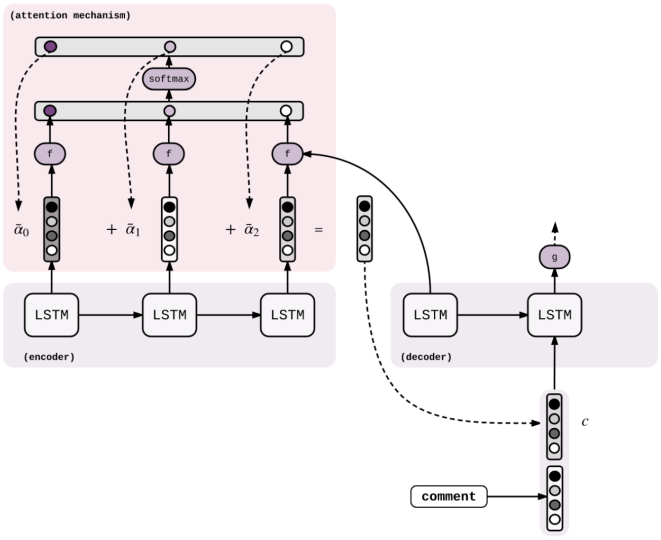

Figure 5: Attention Mechanism's Architecture (Guillaume et al., 2017)

The Seq2Seq model only using the last state of the encoder as input of the decoder, if the information is more complex, it will damage the accuracy, since all the information was compressed in a single vector. Moreover, when the decoder generates the output in one step, it is obvious that some parts of the input sequence are more decisive for some parts of the output sequence. For example, in our case, "How many engine types did Val Musetti use?", when generating the "COUNT" in SQL, "how many" should play a big role compare with other parts from the input.

Attention mechanism using all the encoder outputs and the last state of the decoder (for the first decoder cell, using the final state of the encoder) to calculate the attention score to see which part will play an important role in the current step. Then using the attention score and the last state as an input to calculate the current decoder cell output. It is worth noticing that the attention score is different for each decoder cell since the correlation between each step focusing and the input is different.

The calculation steps (Figure 6 [3]):

1. Using the last decoder state and the output for each encoder cell to calculate each encoder cell score.
2. Softmax each score.
3. Weighted average each score and each encoder output to get the attention score for current decoder cell.

$$\alpha_{t_{old}} = f(h_{t-1}, e_{t_{old}}) \in R \text{ for all } t_{old}$$

$$\overline{\alpha} = softmax(\alpha) \ ,$$

$$c_t = \sum_{t_{old}=0}^{n} \overline{\alpha}_{t_{old}} e_{t_{old}}$$

Figure 6: Attention calculation form (Guillaume et al., 2017)

4.3 Pointer Network

There is a kind of alternative Seq2Seq network: the Pointer network. Instead of transforming one language to another. It picks parts of input as output which can describe input most accuracy.

The idea is from the convex set, i.e., forming a set with the least point which could conclude all the points listed. In the NLP area, it is used to write a summary for an article (Figure 7[6]). In our case, in the "Select" and "Where" part, the network needs to point which attribute and which column is selected based on the query question.

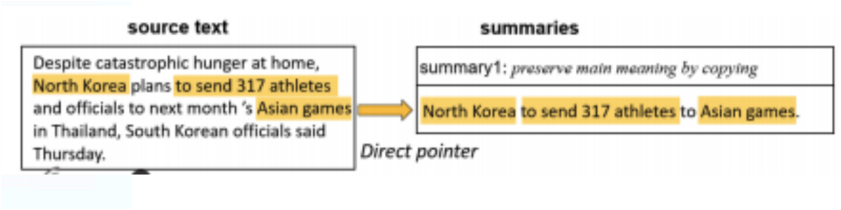

Figure 7: Generating summary by Pointer network (Vinyals et al., 2015)

5 Seq2Sql

Seq2SQL is based on the Seq2Seq model since SQL can be treated as a special language. It has three parts as shown in the graph, these component uses the SQL structure to pure the output space and improve the accuracy compare to directing generating the whole SQL.

5.1 Aggregation classifier

The Aggregation classifier using the Seq2Seq model with an attention mechanism to choose the aggregation operator will be used based on the question, if no aggregation is needed, it would return NULL.

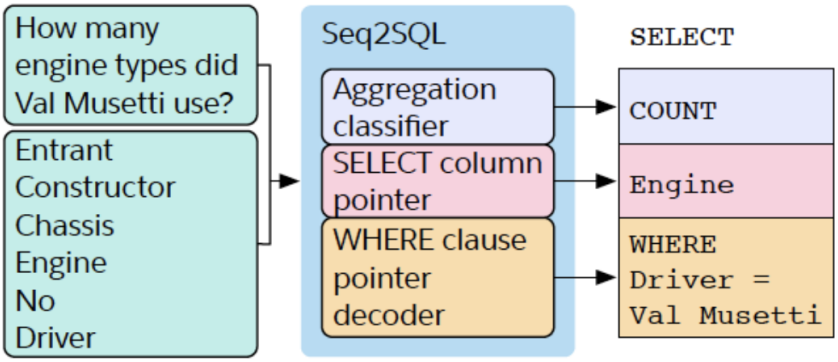

Figure 8: 3 Components in Seq2Sql(Zhong et al., 2017)

5.2 Select Column

The Select Column using the pointer network to find which column will be selected based on the question query and also all the column names. Firstly, using LSTM as an encoder to get each column cell state $h^c_{j,t}$ (8) ; and using K^{sel} to represent question input, which is the sum over the input encodings weighted by the normalized scores. Then computer each column score based on the following form, after that normalize it in order to find the most possible column (10).

$$h^c_{j,t} = LSTM(emb(x^c_{j,t}, h^c_{j,t-1})) \tag{8}$$

$$e^c_j = h^c_{j,T_j} \tag{9}$$

$$\alpha^{sel}_j = W^{sel} tanh(V^{sel} K^{sel} + V_c e^c_j) \tag{10}$$

Both Aggregation and Select part using cross entropy loss during training, which could have a larger update rate since cross entropy loss has a high punished value on incorrect output.

5.3 Where Clause

The Where Clause using the same idea as the Select Column to find which condition can represent the question most precisely. It is also the most challenging part; from the part accuracy, we can see that where part is much lower than the other two since the expression of condition in natural language is more complex. For example, one condition could describe separating in the beginning and plus the ending, or sometimes shorthand could cause ambiguity, e.g., FCB could stand for FC Barcelona or FC Bayern.

Moreover, Where Clause cannot use cross entropy loss, it because that the order has no effect on the correct output, i.e., both “Where A=18 and B=20” and “Where B=20 and A=18” have

$$R\left(q\left(y\right),q_g\right)=\begin{cases}-2, & \text{if } q\left(y\right) \text{ is not a valid SQL query}\\ -1, & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to an incorrect result}\\ +1, & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to the correct result}\end{cases}$$

Figure 9: Reward Function (Vzhong et al., 2017)

the same result. However, cross entropy loss will have different punished values for these two since one of them matches the correct answer, and the other does not. In this case, the reward function 9 is used. Even though using reinforcement learning to optimize the execution result, it only has 2% improvement; as long as the way of generating the Where Clause does not change, the accuracy will not improve in the order of magnitude.

6 SQLNet

6.1 Summary

As stated in approach 1, the seq2seq decoder predicts the next tokens depending on all previous tokens. However, different parts in select and where clauses may not have a dependency on each other. This paper tackles the "order matters" problem using a sketch. The sketch is also an alternate way to solve the issue in Approach 1, the reinforcement learning is often limited, only 2% increment accuracy by employing reinforcement learning.

6.2 Model

The paper provided a sketch-based approach to generate a SQL query from a sketch, to avoid the issue that the order in select and where clause doesn't affect the execution result in the seq2seq model. The SQLNet model used a neural network to predict the continent for each slot. The basic idea of SQLNet is to employ a sketch that misses the core part in the query(select and where clause). Then build dependency is recorded only if the words depend on others, the model only needs to fill in the slots instead of generating the whole query including the grammar of the query and the content with a dependency with all words. The paper introduces sequence-to-set section 6.3 and column attention section 6.4to implement sketch.

In Figure 10(a), The bold token like select and where are the keywords in SQL query. The blue tokens are the slots to be filled. The name after the "$" is the type of prediction. For example, $AGG could be MAX, AVG. The $OP token is a value among $\{=, >, <, \geq, \leq\}$ The $COLUMN is filled with a column name in the table and the $VALUE is from a sub-string of the question.

Figure 10(b) is the dependency graph of the sketch. For example, VALUE1 depends on Column1 and the question but doesn't have a dependency on Column 2 and Value 2. That is the prediction of one constraint is independent with another, and this solves the "order-matters" problem in a sequence-to-sequence model.

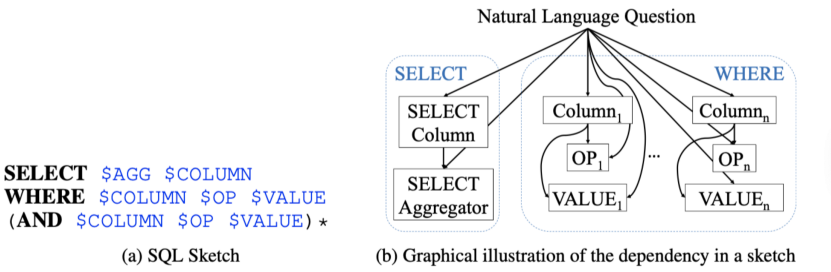

Figure 10: Sketch syntax and the dependency in a sketch(Hwang et al., 2019)

6.3 Sequence to Set

In the where clause, the model predicts a few column names among all columns in the table. In short, sequence-to-set prediction is an idea that predicts which column names to appear.

$$P_{\textbf{wherecol}}(col|Q) = \sigma(u_c^T E_{col} + u_q^T E_Q)$$

Figure 11: Sequence to Set formula(Hwang et al., 2019)

The formula shows a naive approach to compute the probability of a column should appear given the natural language question. Here σ is the Sigmoid function, u_c and u_q are two column vectors of trainable variables. E_{col} and E_q are embedding of the column name and the natural language question. The model uses a bi-directional LSTM to encode E_{col}, E_q(column name and question). The probability is computed for each column name separately. In this way, the probability of including the column name is independent of other columns.

The equation has an issue with E_q, the hidden states of the natural language question, it may not remember the information regarding a particular column. For example, when predicting the column "No." the token "number" is very relevant. Using column attention allows the E_q to reflect the most relevant information in the natural language question for predicting on a particular column.

6.4 Column attention

Column attention compute $E_{Q|col}$ instead of E_q, the concept of column attention is also used in section 7, which is one of the models among the model with highest state-of-art score.

Here E_{col} is the embedding for a column name, H_Q is is a matrix of dimension of LSTM $d \times$ words in the question. H_Q^i indicates the i-th column of H_Q. W is a trainable matrix of size $d \times d$. w is the attention weights.

$$w = \mathbf{softmax}(v) \qquad v_i = (E_{col})^T W H_Q^i \quad \forall i \in \{1, ..., L\}$$

Figure 12: Attention Weight(Hwang et al., 2019)

V_i is related to the importance of word number i in the question regarding the column name. w is a vector with the length of the number of words in question, which records the percentage of importance for each word in the given question.

$$E_{Q|col} = H_Q w$$

Figure 13: Question Embedding using Column Attention(Hwang et al., 2019)

Column i in H_Q is the LSTM hidden output embedding for the i-th token in the question. w is the importance for each token in the question. Therefore, multiples H_Q and w give a new embedding for the question that remembers more information related to the column name. Then replace H_Q with $H_{Q|col}$ the equation in Figure 11 to get the column attention model. This is the core concept in the column attention model.

$$P_{\mathbf{wherecol}}(col|Q) = \sigma(u_c^T E_{col} + u_q^T E_{Q|col})$$

Figure 14: Core Concept in Column Attention(Hwang et al., 2019)

The paper says adding one more layer of affine transformation improves the performance by around 1.5%. Here is the final model:

$$P_{\mathbf{wherecol}}(col|Q) = \sigma((u_a^{col})^T \mathbf{tanh}(U_c^{col} E_{col} + U_q^{col} E_{Q|col}))$$

Figure 15: More Layer Version (Hwang et al., 2019)

6.5 SQLnet model

As mentioned in Section 2, the wikiSQL data set's query only uses one table, therefore, the FROM clause is done automatically. WHERE clause is more complicated than a SELECT clause. X.X explains how does SQLnet predicts the WHERE clause, the hardest part of the task.

6.5.1 Column Slot in Where Clause

As mentioned in Section 6.4, the $P_{\text{wherecol}}(col|Q)$ gives the probability of column col appearing in the SQL query. One approach to decide which column names appear in the query is by setting a threshold $\in$ (0, 1), if $P_{\text{wherecol}}(col|Q)$ >threshold, include the column in the SQL query. A better approach is to use a network to choose a number K from the question. Then chose top K columns with highest $P_{\text{wherecol}}(col|Q)$ scores.

They also set the upper bounds of the columns to include(4). That is 4 ways(include the column or not) + 1 way(no condition in where clause) = 5 ways to choose the column.

Figure 6.5.1 is the formula for the network. In short, the formula gives the K and gets the top K columns with the highest possibility to be included in the query.

$$P_{\#\textbf{col}}(K|Q) = \textbf{softmax}(U_1^{\#\text{col}}\textbf{tanh}(U_2^{\#\text{col}}E_{Q|Q}))_i$$

where $U_1^{\#\text{col}}$ and $U_2^{\#\text{col}}$ are trainable matrices of size $(N+1) \times d$ and $d \times d$ respectively.

Figure 16: Formula for Column Slot in Where Clause (Hwang et al., 2019)

6.5.2 OP Slot in Where Clause

The paper wrote for each OP slot(operator in where clause), there are 3-way classifications, {=, >, <}. U_1^{op}, U_c^{op}, U_q^{op} are trainable matrices of size 3 $\times d$, $d \times d$, $d \times d$ respectively, noted $E_{q|col}$ is on the right, which means the column attention is used to predict the operator.

$$P_{\text{op}}(i|Q, col) = \textbf{softmax}(U_1^{\text{op}}\textbf{tanh}(U_c^{op}E_{col} + U_q^{op}E_{Q|col}))_i$$

Figure 17: Formula for OP Slot in Where Clause (Hwang et al., 2019)

6.5.3 Value Slot in Where Clause

The task is to find a substring of the natural language question to be the value, given the question and column name. The order of tokens in question is also valuable information to fill the value slot. Therefore, the paper uses the seq2seq model with the pointer network.

The paper uses the pointer network with the column attention mechanism and encoder as a bi-directional LSTM. The probability of the next token in VALUE is:

The task can be divided into two parts, the first one is to find the columns, which is very similar to finding the columns in the where clause. The second part is to find the aggression, which is

$$P_{\text{val}}(i|Q, col, h) = \textbf{softmax}(a(h))$$

$$a(h)_i = (u^{\text{val}})^T \textbf{tanh}(U_1^{\text{val}} H_Q^i + U_2^{\text{val}} E_{col} + U_3^{\text{val}} h) \quad \forall i \in \{1, ..., L\}$$

where u_a^{val} is a d-dimensional trainable vector, $U_h^{\text{val}}, U_c^{\text{val}}, U_q^{\text{val}}$ are three trainable matrices of size $d \times d$, and L is the length of the natural language question. Note that the computation of the $a(h)_i$ is using the column attention mechanism, which is similar in the computation of $E_{Q|col}$.

Figure 18: Formula for Value Slot in Where Clause (Hwang et al., 2019)

very similar to the OP slot in the where clause. Here the trainable matrix's size is $6 \times d$ instead of $3 \times d$ ([", 'MAX', 'MIN', 'COUNT', 'SUM', 'AVG'])

6.5.4 Loss Function

The paper gives the loss function for column selection for where clause, the model rewards the correct prediction and published the incorrect prediction, hyper-parameter α is the weight to balance the positive and negative data.

$P_{\textbf{wherecol}}$:

$$loss(col, Q, y) = -\left(\sum_{j=1}^{C} (\alpha y_j \log P_{\textbf{wherecol}}(col_j|Q) + (1 - y_j) \log(1 - P_{\textbf{wherecol}}(col_j|Q)) \right)$$

Figure 19: Formula for Loss Function (Hwang et al., 2019)

Q is the question, y is the indicator(1 if col in the ground truth, 0 otherwise). C is the set of all columns in the table. The model train the sub-model P_{wherecol} by minimize the negative log-likelihood loss.

7 SQLova

7.1 Summary

The NL2SQL Layer of the model is very similar to the third approach. Which uses all the features in Section 6 including sketch, Sequence to set, and column attention. The model takes full advantage of BERT[5] and outperforms 21.59 percent to the SQLNet, which is the test accuracy from 68.0% to 89.6%, which reaches the upper bound in WikiSQL.

The first layer of the model is Table-aware Encoding Layer, which feeds natural language questions and table columns into BERT and outputs the encoded representations. The second layer is the NL2SQL, which takes the encoded representations in the first layer and generates the query.

7.2 Table-aware Encoding Layer

The table-aware encoding layer encodes the natural language and the column names of the table together using table-aware BERT. [SEP] is a special token to separate the natural language question and the column names of the table. The natural language query is encoded as $T_{n,1}, T_{n,2}, ...T_{n,l}$ where L is the number of words in the question. The column names of the tables are encoded as $T_{h_1,1}, T_{h_2,1}$ where$T_{h_1,1}$ stands for the first token of the first column name, and $T_{h_2,1}$ stands for the second token of the first column name.

For example, if the query is "What are the ids where students' family name is Harrison", the table columns are id, family name, "id" is encoded as "id", "family name" is encoded as "family, name". Then use the special token [SEP] to join them, the whole input is encoded as [CLS], What, are, the, ids, where, students', family, name, is, Harrison, [SEP], id, [SEP], family, name, [SEP]. An example in the paper is in Figure 20 [9].

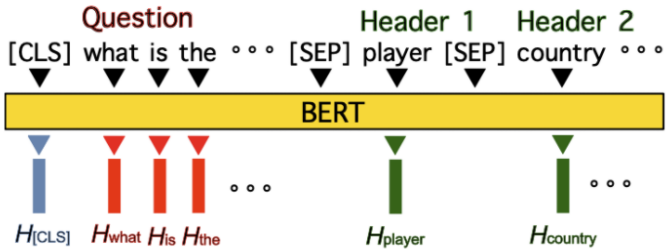

Figure 20: SQLova's Table-aware Encoding Layer (Hwang et al., 2019)

Another input for BERT is the segment id, the paper uses 0 for the question tokens and 1 for the header tokens.

7.3 NL2SQL Layer

NL2SQL LAYER (Figure 21) is a layer on top of the table-aware encoding layer. Syntax-guided sketch and column-attention are also frequently used in this layer. Therefore, in this section, I will introduce the select-column model to compare with the where-column part in the SQLNet(third approach).

The select column task is to choose among all column names in the SQL table and find out which column(s) to put in the SQL query. In Equation X, $\mathcal{W}$ is used to denote trainable matrix. Even though $\mathcal{W}$ has different sizes in different places, the author uses the $\mathcal{W}$ symbol for the ease of understanding the model.

Here D_c denotes the encoding of the c-th column name, and E_n denotes the LSTM 's output for the n-th token in the natural language question. Therefore, after softmax $\boldsymbol{s(n|c)}$, it gets the attention weight for the question token number n, and the column number c. Then in the third line of Figure x, C_c gets a vector for the query question, where it is calculated as the sum of the

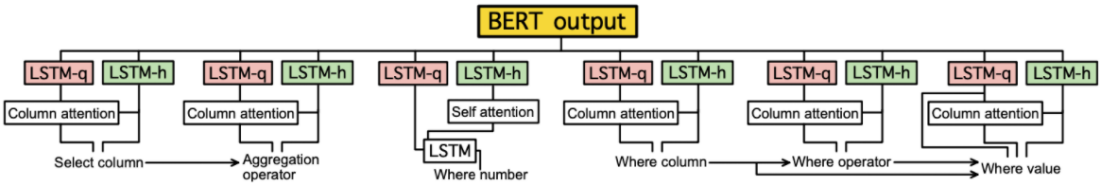

Figure 21: SQLova's NL2SQL Layer (Hwang et al., 2019)

$$
\begin{aligned}
s(n|c) &= D_c^T \mathcal{W} E_n \\
p(n|c) &= \mathbf{softmax}(s(n|c)) \\
C_c &= \sum_n p(n|c) E_n \\
s_{sc}(c) &= \mathcal{W} \tanh([\mathcal{W} D_c; \mathcal{W} C_c]) \\
p_{sc}(c) &= \mathbf{softmax}(s_{sc}(c))
\end{aligned}
$$

Figure 22: Formula of SQLova

$$
\begin{aligned}
w &= \mathbf{softmax}(v) \\
v_i &= (E_{col})^T W H_Q^i \quad \forall i \in \{1, ..., L\} \\
E_{Q|col} &= H_Q w \\
P_{\mathbf{wherecol}}(col|Q) &= \sigma(u_c^T E_{col} + u_q^T E_{Q|col})
\end{aligned}
$$

Figure 23: Formula of SQLNet

attention weight for the n-th token in the question times the LSTM's output for the n-th token in the question for each token in the question.

$\mathcal{W}D_c; \mathcal{W}C_c$ is the concatenation of the two vectors. In SQLnet(third approach) it uses sum instead of concatenation. Then softmax $\mathcal{W}_{sc}(c)$ to get $\mathcal{P}_{sc}(c)$, which indicates the probability of putting c-th column name in the selection clause of the result query.

7.4 Comparison

Figure 34 is the where column clause part in the SQLnet (third approach), since finding the column in where clause is very similar to finding the column in select clause, we use the column clause in SQLnet to compare. In the SQLnet, the embedding for the natural language question for a given column col is denoted as $E_{Q|col}$ and computed as $H_Q w$, where w is the attention weight for each token in the question for a given column name col. The question embedding using attention $E_{Q|col}$ is calculated by the original matrix times the attention weight vector. In SQLova, the question embedding using attention is calculated in the same way, instead of using matrix times the weight vector, it uses a summation operator to get the query embedding using the attention mechanism.

In short, the NL2SQL layer is motivated by SQLNet and has the following key difference.

- As mentioned in the NL2SQL Layer section, when combining two vectors corresponding to the natural language question and the column names, SQLNet uses addition and NL2SQL

uses concatenation instead.

- In SQLNet, different components in SQLNet only share the word embedding, they find sharing the same word embedding vector improves the performance, but in NL2SQL, the layer does not share parameters.
- In SQLNet, when predicting the value slot, the decoder computes the distribution of the next token using a pointer network with the column attention mechanism. NL2SQL train for inferring the start and the end positions of the utterance.
- When finding the Value slot in the WHERE clause in SQLNet, the dependency in the sketch only depends on the column name of the WHERE clause. In NL2SQL, the where-value module depends not only on where-column but also on where-operator.

7.5 Execution-Guided Decoding (EG)

EG is a mechanism during the decoding (SQL query generation) stage, to exclude the non-executable SQL query candidates to get more accurate results.

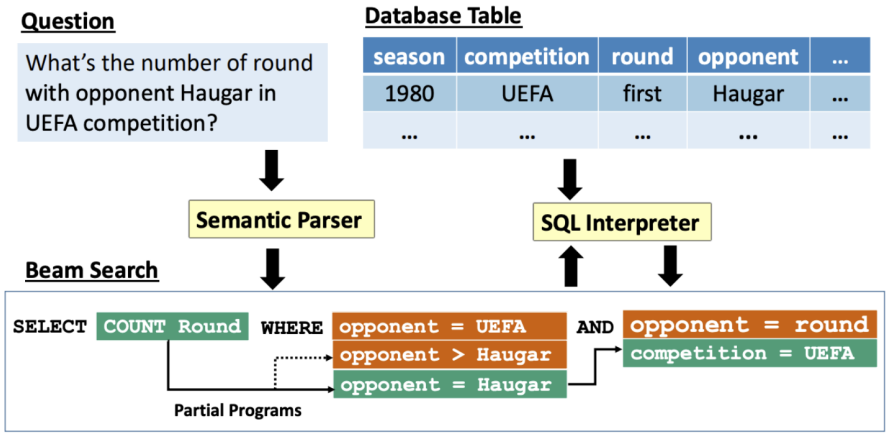

Figure 24: EG Demo (Wang et al., 2018)

An execution-guided(EG)[1] decoder excludes the error queries and empty result queries. Figure 24[1] is an example of using EG, the orange background candidates are excluded. Here, "opponent >Haugar" would yield a run-time error, whereas "opponent = UEFA" would yield an empty result.

EG evaluates partially generated queries and then excludes the candidates that cannot be completed to a correct SQL, an example is in the Select clause, the partially generated query asks the SUM of string values. EG also excludes the queries that generate empty results. The WHERE clause of the result query is generated by selecting the output that maximizes the joint probability estimated from the output of where-number, where-column, where-operator, and where-value modules.

8 Related Work

We did some modifications on the Seq2SQL model, try to improve its accuracy, and see what will happen. Since any modifications during the training part could lead to the different learning speed and verification accuracy, we cannot keep using the original code's training epochs (It executes 50 times before reach the best fitting). In this case, we add an "avoid overfitting" part ((As shown in Figure25) , if the training parts already execute more than 35 times and the last three verifications' accuracy are smaller than the best one, it will stop the training process.

```
#Stop early
if ((training_accuracy[0]>=0.78) or i>=35):

    if ((highest>validation_accuracy_lst[-3]) and
        (highest>validation_accuracy_lst[-2]) and
        (highest > validation_accuracy_lst[-1])):
        print("Braking down at epoch %s" %(i+1))
        break
```

Figure 25: The code could avoid over-fitting

8.1 Changing the word embedding

First, we changed the word embedding, the original code using glove300, and we tried on FastText and Word2vec.

```
#new adding
def load_word_embeddings_fasttext(fname):
    fin = io.open(fname, 'r', encoding='utf-8', newline='\n', errors='ignore')
    n, d = map(int, fin.readline().split())
    data = {}
    for line in fin:
        tokens = line.rstrip().split(' ')
        data[tokens[0]] = np.array([float(x) for x in tokens[1:]])
    return data
```

Figure 26: The new method for read FastText

In Figure 27, we can see that Glove and FastText have the same level of accuracy in verification training; however, the FastText triggers the stop early mechanism. Meanwell, Word2vec has a lower accuracy than these two.

In the test result, the original model, with Glove embedding, still got the highest accuracy in both logic and execution. The model with FastText is very close to it, with only 0.02% disparity, and the Word2vec has 6% difference from these two. It verifies the conclusion that we inferred from the verification curve.

This is because both Word2vec and FastText were based on the local training, which uses a

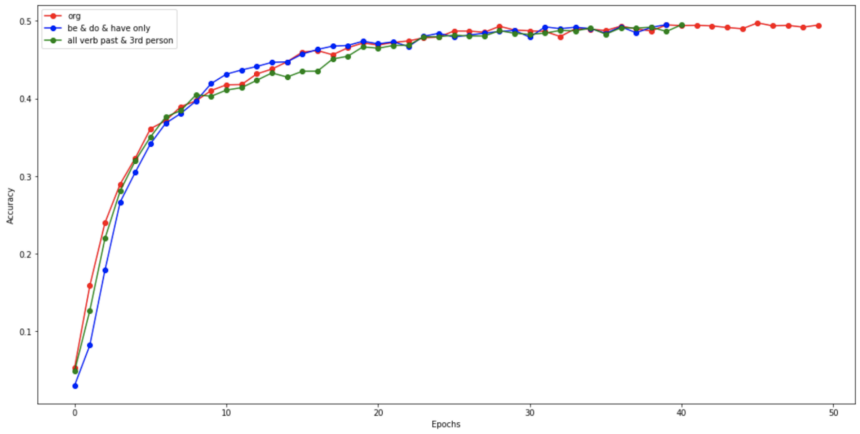

Figure 27: The accuracy of using different word embedding

sliding window to scan the training sentence. It focuses on the connection between the word beside each other, while it loses the global information. Compare with these two, glove was based on global training.

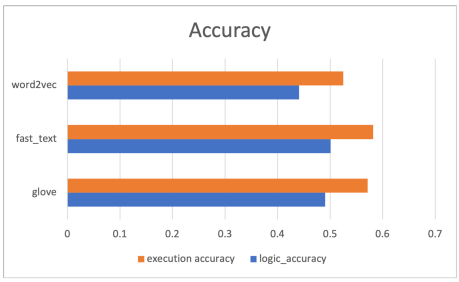

Figure 28: Test result of different word embedding

type	logic_accuracy	execution accuracy
glove	0.5011	0.583
fast_text	0.5009	0.5818
word2vec	0.4408	0.5248

Figure 29: Test result of different word embedding

However, FastText introduces sub-word technology during training, so it could dispose of non-appearance words, which complements some part of losing global information defect. Therefore, the accuracy of the model using FastText could reach the same level as glove.

8.2 Lemmatization on input questions

We also do some lemmatization on the input questions to see whether a good pre-processing could help with the accuracy ((As shown in Figure 30).

1. All verbs $\Longrightarrow$ base form
   We changed all the verbs to their base form, with our unexpected, the accuracy does not increase, instead of decreased to 55%. Therefore, we read the generating queries by the

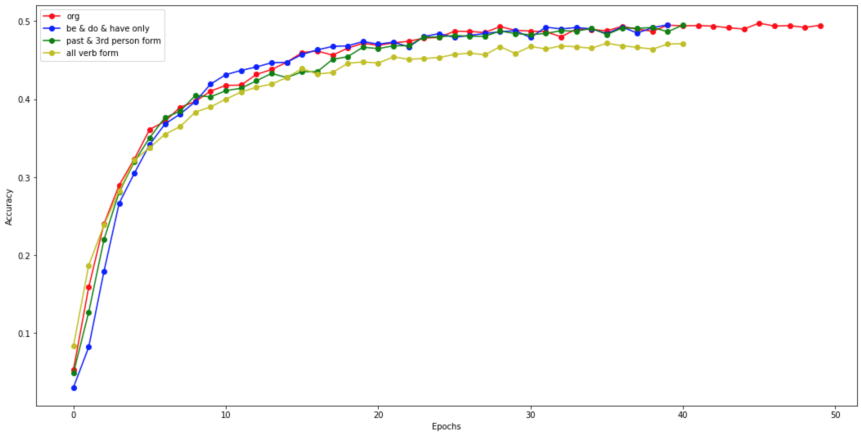

Figure 30: The accuracy of lemmatazion on different verb

original code and the modified code, we found that in several cases, the column name was ending in '-ing', e.g., "Holding" and "Winning". As all the verbs in their base form, the query could not locate in the correct column.

2. part of verbs $\Longrightarrow$ base form
   After it, we only changed all the 3rd-person form verbs and past form verbs (As shown in Figure 31) to the base form to see the accuracy change. In this case, the accuracy kept the same level as the original code, and it let the algorithm stop early.

```
def lemmatize_verb(lemmatizer,sentence):
    lemmatizer = WordNetLemmatizer()
    res=[]
    for word, tag in pos_tag(sentence):
        if tag=="VBD" or tag=="VBP" or tag=="VBZ":
            res.append(lemmatizer.lemmatize(word,wordnet.VERB))
        else:
            res.append(word)
    return res
```

Figure 31: Lemmazation the input question

3. be do have
   After we read the input text, we found that most of the input was in the question form which always changes in "be, do, have" words. Plus, there is a tiny column named end with "-ed". So, we only edited on "be, do, have" words (As shown in Figure 32), which got a 0.5% improvement in testing compared to part 2.

```
def lin_processing(lst):

    if 'has' in lst:lst[lst.index("has")]="have"
    if 'had' in lst: lst[lst.index("had")]="have"
    if 'were' in lst: lst[lst.index("were")]="be"
    if 'was' in lst: lst[lst.index("was")]="be"
    if 'did' in lst: lst[lst.index("did")]="do"
    if 'does' in lst: lst[lst.index("does")]="do"
    if 'am' in lst: lst[lst.index("am")]="be"
    if 'is' in lst: lst[lst.index("is")]="be"
    if 'are' in lst: lst[lst.index("are")]="be"

    return lst
```

Figure 32: Only edited on be & do & have

We can conclude that a good pre-processing cannot help with a big increase in accuracy since the algorithm still generates the query in the same way, which does not have enough power to handle the complex expression, however, it could lead to an early stop for saving time. And also, if we only modified part of the input (the NL question), it could lead the network to lose matching with un-modified parts (the column name).

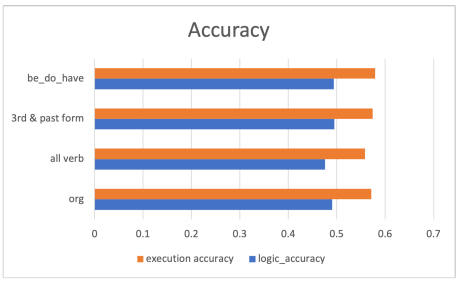

Figure 33: Test result of different lemmatization

type	logic_accuracy	execution accuracy
glove	0.4904	0.5714
fast_text	0.5009	0.5818
word2vec	0.4408	0.5248

Figure 34: Test result of different lemmatization

9 Conclusion

Natural language interfaces (NLI) is a research area at the intersection of natural language processing and human-computer interactions is a hot topic in recent days with the help of machine learning and deep learning. It seeks to provide means for humans to interact with computers through the use of natural language. Processing natural language into executable SQL query is such a meaningful task that attracts much interest from both academia and industry. The test execution accuracy jumps dramatically from 35.9% by Victor Zhong in 2017 to 91.8% by Pengcheng in 2019.

In 2017, Victor Zhong proposed the first approach to generate a query from a question, which only has an accuracy of 35.9%. In the same year, Victor Zhong proposed another approach

using reinforcement learning with an accuracy of 59.4%. SQLNet proposed Sequence to Set and Column Attention to replace the reinforcement learning approach to solve the issue of order-mater problem. Increase the accuracy by 8.6%(59.4% to 68.0%). SQLova(2019) uses the same idea as SQLNet uses, and get a huge advantage of using BERT, gets 18.2% increment and reaches 86.2%. With the Execution-Guided Decoding mechanism, it improves 3.4% of accuracy and reaches 89.6%. X-SQL proposed from Microsoft reaches a state-of-the-art accuracy of 91.8%. But the detailed idea hasn't been public. From SQLova's author believes the performance of SQLova is 2% higher than human and there are issues with the Wiki SQL dataset. The performance of SQLova is near the upper bound of Wiki SQL.

By reading all the approaches and concepts, we were able to understand the concept of complex models step by step. Furthermore, we saw how the researchers share their knowledge and making progress step by step.

References

[1] Chenglong wang, kedar tatwawadi, marc brockschmidt, po-sen huang, yi mao, oleksandr polozov, rishabh singh. 2018. robust text-to-sql generation with execution-guided decoding. corr, abs/1807.03100. https://arxiv.org/abs/1807.03100.

[2] Cover page picture. https://towardsdatascience.com/natural-language-to-sql-use-it-on-your-own-database-d4cd5784d081.

[3] Guillaume genthialr. 2017.seq2seq with attention and beam search. https://guillaumegenthial.github.io/sequence-to-sequence.html.

[4] Ilya sutskever, oriol vinyals, quoc v. le. 2014. sequence to sequence learning with neural networks. corr, abs/1409.3215. https://arxiv.org/abs/1409.3215.

[5] Jacob devlin, ming-wei chang, kenton lee, kristina toutanova. 2018. bert: Pre-training of deep bidirectional transformers for language understanding. corr, abs/1810.04805. https://arxiv.org/abs/1810.04805.

[6] Oriol vinyals, meire fortunato, navdeep jaitly. 2015. pointer networks. corr, abs/1506.03134. https://arxiv.org/abs/1506.03134.

[7] Sepp hochreiter, jurgen schmidhuber. 1997. long short-term memory. https://www.bioinf.jku.at/publications/older/2604.pdf.

[8] Victor zhong, caiming xiong, richard socher. 2017. seq2sql: Generating structured queries from natural language using reinforcement learning. corr, abs/1709.00103. https://arxiv.org/abs/1709.00103.

[9] Wonseok hwang, jinyeong yim, seunghyun park, minjoon seo. 2019. a comprehensive exploration on wikisql with table-aware word contextualization. corr, abs/1902.01069. https://arxiv.org/abs/1902.01069.