SeqGenSQL - A Sequence Generation Model for Structured Query Language

Ning Li
UC Berkeley
louisli@berkeley.edu

Bethany Keller
UC Berkeley
bethany.keller@berkeley.edu

Abstract

Natural language translation to SQL is an active research area in natural language processing. In this paper, we use the largest annotated natural language to SQL transaction data set WikiSQL (Zhong et al. (2017)). Most of the recent research for this task has been focused on assembling predicted elements to guarantee syntactic correctness. Sequence generating models are not very popular due to the randomness of word generations. In this paper, we revisit sequence generation for this task using a pre-trained state of the art sequence generation model T5 (Raffel et al. (2019) with carefully designed question augmentation, data augmentation and model modification. As a result, we have built a new state of the art weakly supervised model (SeqGenSQL, 90.3% execution accuracy), exceeding the previous state of the art weakly supervised model by more than 6% in execution accuracy (83.9% Min et al. (2019)) on the test data set. The final result of SeqGenSQL is comparable to supervised models using annotated logical forms and execution guidance inference. SeqGenSQL generates ready-to-execute SQL statements in one step, providing an elegant approach for structured language generation tasks with accuracy.

1 Introduction

Our work focuses on using a pre-trained state of the art sequence generation model T5 with carefully designed question augmentation, data augmentation and architecture modification.

We started T5-small as base model with including natural question and column headers as input features, SQL statement as output features. We then experimented with including additional features like data types and data samplings. Test results showed that data types and data sampling also contain information that could be learned by the model.

Even with 56,355 training samples, it's still arguably not enough data for large natural language processing models. We designed data augmentation techniques like replacing the select column or the condition value in the where clause. To further expand the data set, we trained a reverse trainer model to get access to practically unlimited number of training data. In our experiments, we generated 250K more silver data for training.

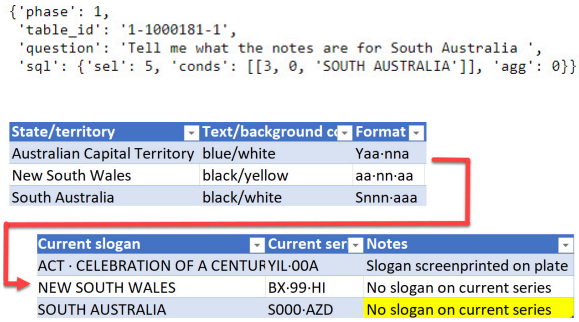
As an attempt to reduce hallucination and encourage extraction (e.g. column names from a natural language question), we implemented a gated extraction layer which decides whether the model should extract (from the natural language question) or generate from the decoder.

There are two leader boards for the WikiSQL data set: weakly supervised (without using logical form during training) and supervised (with logical form during training). [1] On the supervised leader board, there are two results: those with execution guided inference and those without execution guided inference. The previous state of the art weakly supervised model (HardEM, Min et al. (2019)) achieved 83.9% execution accuracy on the test data set. On the supervised model leader board, HydraNet (Lyu et al. (2020)) achieves 89.2% execution accuracy without execution guided inference on the test data set and 92.2% execution accuracy with execution guided inference.

2 Background

WikiSQL contains 56,355 training examples, 8,421 dev examples and 15,878 test examples. This data set includes natural language questions, table ids, annotated SQL logical form and data for each table.

[1] https://github.com/salesforce/WikiSQL



Logical form is annotated by hand and includes an index for a select column, an index for an aggregation function, indices for a where condition column, and indices for where condition operators and values.

The output is a complete SQL statement:

SELECT function(Column)
FROM (Table ID)
WHERE [Condition1, condition2, ...]

SQL statements in the WikiSQL data set have the following properties:

1. One select column
2. One select column aggregation function
3. One table for each question
4. Multiple “where” conditions

Most of the research done for this task uses a slot filling approach: predicting columns, functions and conditions separately then assembling the prediction into a final SQL statement. The work done by Hwang et al. (2019) showed that fine-tuning pre-trained models can provide significant improvement in many areas. In X-SQL (He et al. (2019)), a BERT/MT-DNN based fine-tuned model has shown significant improvement in both logical form prediction and execution accuracy as compared to previous models. Another state of the art model (supervised), Hybrid Ranking Network (Lyu et al. (2020)), is also based on a BERT/RoBERTa pre-trained model and achieved 92% execution accuracy using annotated logical form and execution guided inference - predictions returning empty results will be dropped and the next most probable prediction is chosen.

Historically, sequence generation models like Seq2Seq, (Zhong et al. (2017)) which use a sequence generating decoder architecture to generate SQL statements, had challenges ensuring correct SQL syntax due to ”hallucination” (generating data that does not actually exist) from deep learning models.

In this paper, we argue that with state of the art sequence generation models like T5 and with carefully designed question augmentation and data augmentation, a sequence generating model can overcome this ”hallucination” challenge and generate SQL statements in one step with precise syntax and good execution accuracy.

T5 is also based on transformer (Vaswani et al. (2017)) architecture with additional layers as a decoder for sequence generation which treats all NLP tasks as sequence to sequence generation. It is pre-trained on a data-rich task before being fine-tuned on a downstream task. To establish a baseline, we used T5-small as a base model and followed a commonly adopted practice of combining the natural language question and table columns as input with SQL statements as output. (Zhang et al. (2019))

```
<bos>Natural question<sep>
table-id<sep>col1<sep>col2
<sep>...coln<eos>
```

The labels are executable SQL statements[2] , like

SELECT [notes] FROM [1-1000181-1]
WHERE [state/territory] = ’south australia’

3 Methods

3.1 Question Augmentation

Besides column names, we believe that data types also contain information to be learned for sequence generation. For example, aggregation functions such as avg and sum can only be applied to numeric data columns. Based on this belief, we construct the new input to include data types.

```
<bos>Natural question<sep>table-id
<sep>col1<sep>type1<sep>col2
<sep>type2<sep>...coln<eos>
```

[2] All database object names are quoted according to SQL syntax.

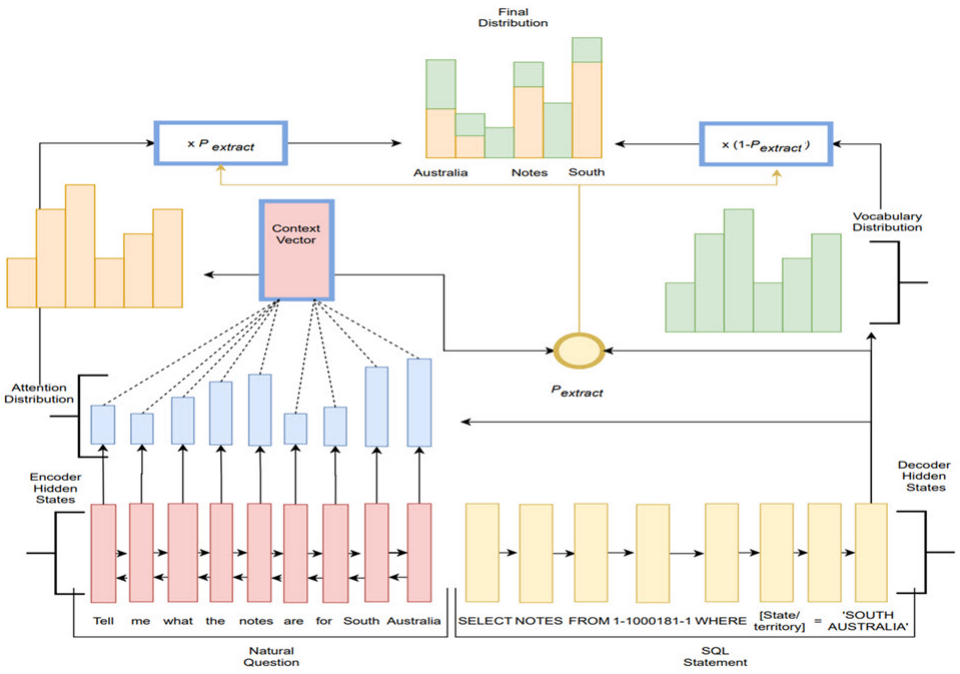


Figure 1: Gated Extraction Layer

Additionally, we borrow from the intuition of how humans write SQL statements - we examine the first few records to learn about the patterns and data format to guide us to write an informed statement. This technique further enhanced the input to include the first few samples from each table. We experimented with sampling 1 row, sampling 2 rows and sampling 3 rows. Simply sampling 1 row showed significant improvement while sampling 2 or 3 produced better outcomes as shown in Table 1.

3.2 Data Augmentation

To augment training data, we decided to replace the select column or change the where condition value from the provided table. In order to do that we randomly select a new column, then replace the column name in the original question with the new selected column name.

```
Old Question: Which score has a record of 58-15-8?
New Question: Which attendance has a record of 58-15-8?
SQL: SELECT [Attendance] FROM [2-13034488-9] WHERE [Record] = '58-15-8'
```

To replace the where condition value, we apply the same technique - randomly select a value from the provided column and replace the old value with the new value in the original question.

```
Old Question: Which Name has a 3rd run of 36.200 (1)?
New Question: Which Name has a 3rd run of 36.457 (2)?
SQL: SELECT [Name] FROM [2-18603914-7] WHERE [3rd run] = '36.457 (2)'
```

3.3 Reversed Trainer Model

We also experimented with a ”reversed trainer” model - flipping input and output for our pre-trained T5 model and fine tuning it (Artetxe et al. (2019)). This reversed trainer model takes a SQL statement as input and predicts a natural language question. We train the ”reversed trainer” model for 20 epochs, then we randomly generate SQL statements from the tables provided, pass generated SQL statements through the reversed model and predict the natural language questions (silver data). Finally, we use this silver data to further fine tune the primary model. In our experiment, we generated 250,000 training samples in addition to the original 56,355 training samples.

Reversed Trainer Model

```
trainerModel = Finetune T5 using ([SQL,Question]) pairs
while:
  selColIdx = random(len(table columns))
  aggFuncIdx = random(len(aggFuncList))
  numCondition = random(4)
  for cond in numCondition:
    condColIdx = random(len(table columns))
    condValueIdx = random(len(table rows))
  trainerSQL = Compose SQL statement using
               selColIdx, aggFuncIdx and condString
  trainerQuestion = trainerModel.generate(trainerSQL)
```

3.4 Gated Extraction Network

Like other deep learning models, T5 also hallucinates and generates completely new words:

```
In which country is the city of Netanya?
Pred: select [country] from [1-14937957-1] where [city] = 'netheranya'
True: select [country] from [1-14937957-1] where [city] = 'netanya'
```

To reduce hallucination and encourage extraction, we implemented a gated extraction T5 network. Similar to a Pointer Generation Network (See et al. (2017)), we implemented a cross layer attention layer between the encoder(H_{enc}) and decoder (H_{dec}). Then we created a gate layer from the attention layer to control whether the output should be generated by the decoder or extracted from the encoder.

The cross layer attention layer is implemented the same way as the T5 cross attention layer where the score is the product of H_{enc} and H_{dec}:

$$q = linear(H_{dec})$$
$$k = v = linear(H_{enc})$$
$$Score = q * v$$

The context is then calculated using $Score$, H_{dec} and $softmax$ function:

$$Context = FF(softmax(Score) * v)$$

The final gate is a probability based on the hidden state of the decoder and $Context$ wrapped inside a sigmoid function:

$$lnorm_{con} = LayerNorm(H_{dec})$$
$$lnorm_{Context} = LayerNorm(Context)$$
$$P_{ext} = sigmoid([lnorm_{con}, lnorm_{Context}])$$

The final step is to merge both extraction and generation together using element wise operation:

$$O_{final} = (1 - P_{ext}) * O_{gen} + P_{ext} * O_{ext}$$

3.5 Execution Guided Inference

Even with slot filling models, it's challenging to generate perfect SQL prediction. To improve the prediction, Text2Sql (Chenglong Wang (2018)) introduced execution guided inference. Execution guided models send generated SQL statements to the SQL database engine and make adjustments if the database engine returns run-time errors or an empty result. While we are not convinced an empty result should be considered an error in practice, we did experiment with beam search using run-time error during execution.

To apply execution guided inference, we used beam search during inference to generate multiple output sequences for each question. Then, we sent each output sequence to the SQL database engine. If the SQL database engine returned a run-time error, we dropped the current sequence and tried the next output sequence.

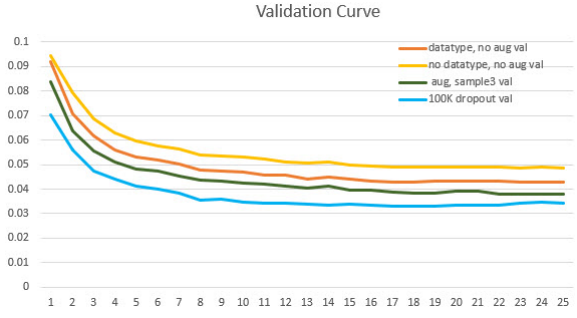
Execution Guided Inference

```
predicts = Predict 3 SQL statements
finalOutput = []
for seq in predicts:
    try:
        run seq in SQL database engine
        finalOutput.append(seq)
        break
    finally:
        continue
```

The top performing models on the leader board applying execution guidance are getting significantly greater improvement in execution accuracy than SeqGenSQL. Error analysis shows SeqGenSQL only had only roughly 30 predictions using the dev data set with execution errors. Since SeqGenSQL is already making such accurate predictions, execution guidance may not provide much additional value.

4 Results

Using T5-small, our baseline model achieved 80.0% execution accuracy. With additional question augmentation like adding data types and data sampling, execution accuracy improved to 86.7%, which exceeded the previous state of the art weakly supervised model (84.4%, Min et al. (2019)).



Using a reversed trainer model and input token drop-out (randomly drop one masked token during training), we gained a further 1.6% improvement.

Our final model, built on a pre-trained T5-base (which contains 3x the parameters than T5-small) with data type and data sampling, trained on 56,000 original training data samples and 250,000 silver training data samples, achieved 90.5% execution accuracy on the dev data set and 90.2% execution accuracy on the test data set. This model exceeds all models on the weakly supervised leader board and all models without execution guidance on the supervised leader board. SeqGenSQL lands 3rd place overall on the combined leader board.[3]

5 Analysis and Discussion

Sequence generation simplifies the SQL statement generation process and achieves a one step prediction for this task with surprising results. During error analysis we identified mistakes made by SeqGenSQL: hallucination, multilingual text, ambiguous questions, missing information and data issues.

5.1 Hallucination

Even with a gated extraction network, hallucination still happens. In the following example, SeqGenSQL generated a new value "1962-001a" in the where condition instead of extracting the word "1962-011a" from the question:

Question:
How many alt names does 1964-011a have?
Prediction:
select count([alt name]) from [1-12141496-1]
where [id] = '1962-001a'

5.2 Multilingual Text

WikiSQL is a crowd sourcing data set and contains multiple languages. T5 was trained on only English data. Therefore, the T5 tokenizer cannot interpret non-English characters:

How many people live in 铅山县?

5.3 Ambiguity in Questions

Some questions contain words that are ambiguous. For example, the word "total" in the following question can be interpreted as a sum function in SQL:

Question:
What is the total brup for the team?
Gold Label:
select [brup] from [1-18064020-21]
where [name] = 'total'

5.4 Missing Information

Some of the questions simply didn't provide sufficient information to compose an accurate SQL statement. In this example, the "gold label" (human-generated label for comparison) uses the "hardcover" column while SeqGenSQL chose to use "paperback". The difference between these two columns cannot be identified from the question.

Question:
How many publishers put out isbn 193700788x?
Gold Label:
select count([publisher]) from [1-16907214-1]
where [hardcover]='isbn 193700788x'
Prediction
select count([publisher]) from [1-16907214-1]
where [paperback] = 'isbn 193700788x'

5.5 Data Issues

There are cases where gold labels don't match the questions. In some of these cases, SeqGenSQL predicted correctly in manual verification, even though they don't match gold labels:

Question:
Which places have points larger than 10?
Gold Label:
select min([points]) from [2-10301911-6]
where [place] > '10'
Prediction
select [place] from [2-10301911-6]
where [points] > '10'

Even though some of these errors (data issues), might not be solved with a better model, we still consider the following things that could be solved by improving the model. For example:

1. Larger base model

There are 5 pre-trained T5 models available. We experimented with T5-small and T5-base and

[3] https://github.com/salesforce/WikiSQL

Model	Dev execution accuracy
T5 Baseline (T5-small)	80.0
+datatype	84.1
+datatype+augmentation	84.1
+datatype+1sample	86.0
+datatype+2samples	86.5
+datatype+3samples	86.8
+datatype+3samples+50K silver data	88.3
+datatype+3samples+50K silver data+gated extraction	**88.5**

Table 1: SeqGenSQL Results

Model	Dev execution accuracy	Test execution accuracy
Supervised		
HydraNet + EG (Lyu et al. (2020))	92.4	92.2
X-SQL + EG (He et al. (2019))	92.3	91.8
EG Guided Decoding with BERT-Base Uncased (Guo and Gao (2019))	91.1	90.1
HydraNet (Lyu et al. (2020))	89.1	89.2
X-SQL (He et al. (2019))	89.5	88.7
Weakly supervised		
HardEM (Min et al. (2019))	84.4	83.9
SeqGenSQL(T5-base + 250K silver data)	**90.5**	**90.2**
SeqGenSQL + EG	**90.8**	**90.3**

Table 2: WikiSQL Leader board

observed improvement in accuracy using larger model. We assume larger base models could provide even more improvement.

2. Improve the gated extraction layer
As we observed in error analysis, there is still a small amount of hallucination happening. Further investigation and adjustment could be done to improve execution accuracy in this area.

3. Include more sampling data
Our tests showed more sampling data can improve accuracy. In most of our tests we used 3 samples in input. We could experiment further with a higher number of samples and observe whether it does/does not improve the model.

6 Conclusion

Previously, sequence generation hasn't been a popular choice for structured language generation due to syntactic challenges. Our paper shows that with a pre-trained cutting edge sequence generation model and carefully designed question augmentation, data augmentation and model modification, sequence generation can achieve great results even without execution guided inference and slot filling techniques. Our sequence generation model is able to predict SQL statements in one step with 90.2% execution accuracy.

References

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2019. An effective approach to unsupervised machine translation.

Marc Brockschmidt Po-Sen Huang Yi Mao Oleksandr Polozov Rishabh Singh Chenglong Wang, Kedar Tatwawadi. 2018. Robust text-to-sql generation with execution-guided decoding.

Tong Guo and Huilin Gao. 2019. Content enhanced bert-based text-to-sql generation.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-sql: reinforce schema representation with context.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid ranking network for text-to-sql. Technical Report MSR-TR-2020-7, Microsoft Dynamics 365 AI.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard em approach for weakly supervised question answering.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

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

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