DataBase Systems 2020
Project2 TypeSQL

AM DS1190015

Apostolos Papatheodorou

In this report, it is presented TypeSQL, a sequence-to-sequence model, that attempts to convert natural language sentences into SQL queries. TypeSQL heavily relies on previous Seq2Seq models like Seq2SQL and SQLNet and by introducing some novel techniques, has managed to outperform them. Below are analyzed the structure and the techniques of this model.More details about experiment results are in Results.pdf.

TypeSQL

Introduction

TypeSQL is a sequence-to-sequence model that takes as input natural language sentence and converts it to the corresponding sql queries. It is an expansion of SQLNet. The way It is improving the previous results is by assigning a type to each query word. These types may come from a Knowledge graph, the table's schema, or from relation's entries(optionally).

In this report, it will be presented the TypeSQL model, its main competitors SQLNet and Seq2SQL, and it will be analyzed the code structure and relative experiments as well.

Seq2SQL & SQLNet

Because TypeSQL heavily relies on SQLNet it is necessary to introduce first the previous works on this topic SQLNet and Seq2SQL.

Sketch vs *Reinforcement Learning*

One of the main adversities of solving a text-sequence to SQL-sentence is that a query clause could have many many equivalent clauses. This issue is known as the ***order-matter problem*** and it may impede the training process as the model is sensitive to the choice of the training set. The usual way to confront this phenomenon is by introducing reinforcement learning.

SQLNet and Seq2SQL adopt two different approaches so as to bypass this issue. SQLNet uses a sketch-based approach (which is suitable when the order of words is not important), while Seq2SQL uses a reward-oriented technique, in other words, it is trained with supervising learning for a period of time and after that is uses a reward strategy further improve the accuracy.

Both models are using the WikiSQL dataset.

WikiSQL

Before starting some details about the dataset.

WikiSQL dataset is large enough for training and contains queries based only on the natural language and the table schema without relying on the table's content. Also training, dev, and test set do not share tables (scalability and generalization properties). This also means that predictions are made only for SELECT and WHERE clauses and not for FROM keywords

Other assumptions of WikiSQL is that the columns' names have meaningful descriptions, each query in WHERE clauses have the format of [COLUMN OP VALUE]and OP belongs to [<, =, >, ≥, ≤] set.

```
-------------  Start Training  ----
train data 56354 table len 18585
val.  data  8420 table len 2716
test  data 15877 table len 5230
```

SQLNet

SQLNet avoids the classical sequence to sequence structure by utilizing a sketch based on the SQL syntax. This means that the model have only to fill in some slots which are created based on the classical SQL syntax. In this way, it achieves 61.5% acc, on exact query match of and 68.3% acc. on result-match (WikiSQL test set).

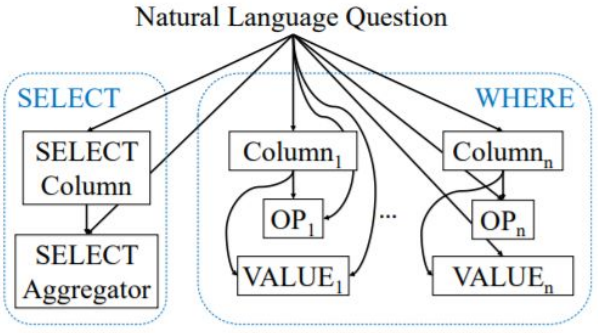
WHERE Clause

The most demanding part of the models is to generate the where clause correctly. Due to the nature of the Where clause, every next output depends on the previous one. Thus, many constraints may be correlated with others or totally independent. To deal with issue SQLNet introduces two new mechanisms *sequence-to-set* and *column attention*. The former concerns the prediction of an unordered set of constraints instead of an ordered sequence whereas the latter to detect dependencies relationships.

Here we will focus on *column attention* as TypeSQL uses it extensively.

Dependencies

As it seems in the figure below, the sketch, by construction is able to capture $Slot dependencies. Therefore the predictions of a $Slot are based only on the input and the relative $Slots. Another of the advantages of sketch approaches which facilitate training is that the model avoids to learn and predict grammar syntax.



(b) Graphical illustration of the dependency in a sketch

Figure-b shows the dependencies of each slot $AGG depends on

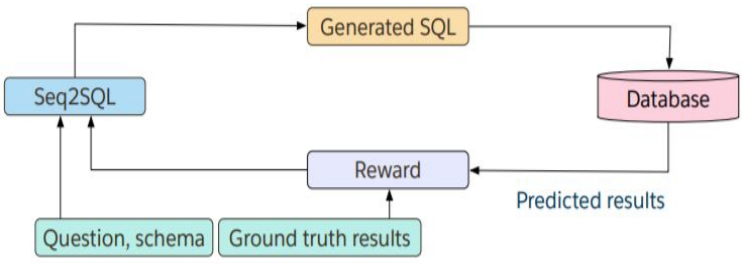
$COLUMN and VALUE1 on COLUMN1 respectively) whilst at the same time, $AGG and VaLUEn are uncorrelated.

Column attention

As all Seq2Seq models, it is plausible that the model will not remember the information useful in predicting a particular column name. The attention mechanism facilitates the embedding to reflect the embedding reflect relevant information in each question. Attention weight are calculated for each sentence and it is used during training.

Seq2SQL

Seq2SQL as SQLNet tries to solve the problems of translating NLQuestions to SQL queries. It takes also advantage of the standard SQL format in order to prune the output space. It consists of three components (AGG, SEL,COND). However, in contrast with SQLNet it takes into consideration the order matter problem and uses Reinforcement learning to make predictions for WHERE clauses.



From the picture, it is clear that the model takes as input the NLQuestion and the table's schema, it produces the output which is executed against a database. The result of the execution is utilized as the reward to the RL algorithm.

The three components of Seq2SQL are:

- The first component is the aggregation operator (if it is not provided null operation is used).
- The second component is for SELECT and identifies the right columns.
- Finally the third component for WHERE clauses.

The first two components are supervised using cross-entropy loss, whereas the third generation component is trained using policy gradient to address the unordered nature of query conditions.

Evaluation

Finally before present the TypeSQL it is necessary to present the format which is used in all the three models.

- N denotes the total number of examples
- Nex the number of queries that, when executed, result in the correct result (Accex = Nex/N)
- Nlf number of queries has exact string match with the ground truth query used to collect the paraphrase(Alf = Nlf N).

Accex and Alf are used concurrently because:

Accex

it is that it is possible to construct a SQL query that does not correspond to the question but nevertheless obtains the same result.

Queries

SELECT COUNT(name) WHERE SSN= 123 and
SELECT COUNT(SSN) WHERE SSN= 123
They may produce the same result.

AccIf

It may incorrectly penalize queries that achieve the correct result but do not have an exact string match with the ground truth query. Due to these observations, we use both metrics to evaluate the models.

SELECT $AGG $SELECT_COL
WHERE $COND_COL $OP $COND_VAL
(**AND** $COND_COL $OP $COND_VAL)*

TypeSQL

TypeSql similar to SQLNet confronts the sequence-to-SQL query problem as a slot filling task and capturing dependencies among different attributes. Furthermore, it utilizes extra information and assigns each word a type as an entity from either the knowledge graph, a column, or a number.

During this procedure, the model tokenizes each input sentence and assign it to each word a type. This type could be a column if the token appears in a table's schema, a number from [Integer, Float, Date, Year], or an entity from [Person, Place, Country, Organization, Sport] if the token is an entity. If tables content is available it can ve used and improve even more its performance.

Methodology

The procedure of producing the SQL query from a natural sentence consists of the following steps:

1. Preprocessing the question inputs by type-entity recognition.

2. Encoding of question words/ types and tables' column names separately.

3. Predicting the values for the slots in the SQL sketch.

Code-Implementation

To see the results of the experiments go to **Results.pdf** or run the **Results.ipynb**. In those files are the accumulated the outputs from all the above models (TypeSQL, SQLNet, and Seq2SQL), their difference, and the improvements.

Very-Briefly-Results

We run the code for the three $Slots *MODEL_COL, MODEL_SEL, MODEL_OPVAL* and we display their accuracy on test set. As it was mentioned before the WHERE clauses have the worst result.

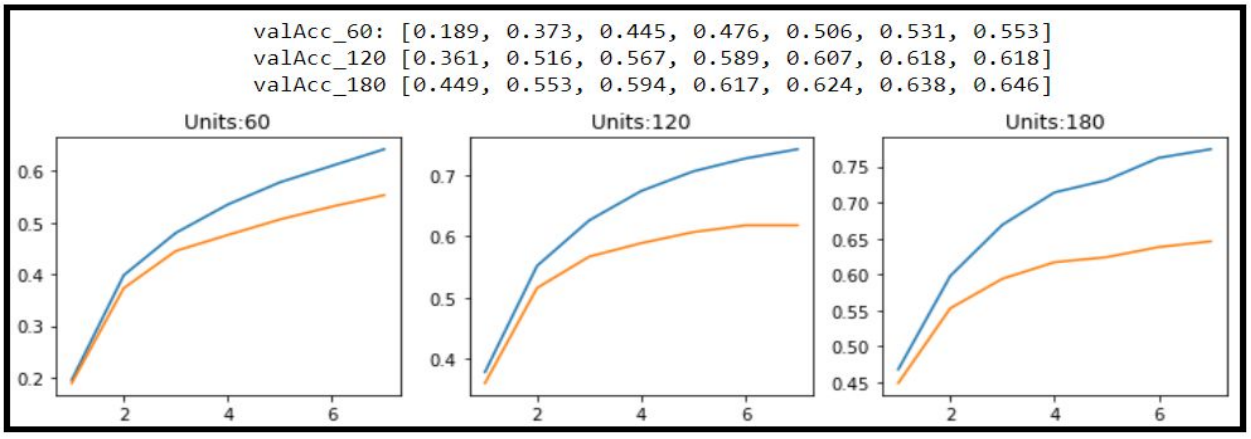
DEV ACCURACY (epochs=7)

Model	Agg	Select	Where
Seq2SQL	92.3%	84.5%	44.4%
SQLNet-Col.Atten.	89.8%	89.0%	53.8%
SQLNet-Col.Atten-Embed.	89.8%	.91.2%	62.9%
TypeSQL-Cont.0.	**89.8%**	**92.5%**	**75.5%**
TypeSQL-Cont.1	**90.0%**	**91.9%**	**86.7%**

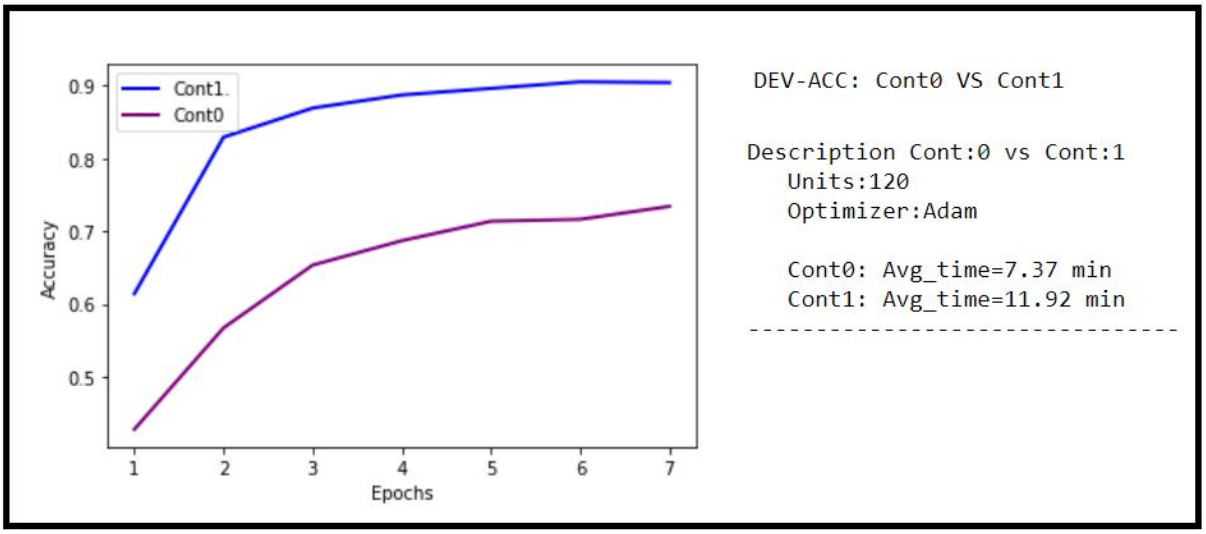
Test ACCURACY (epochs=7)

Model	Agg	Select	Where
Seq2SQL	92.3%	84.3%	45.0%
SQLNet-Col.Atten.	90.0%	88.0%	53.4%
SQLNet-Col.Atten-Embed.	90.0%	90.0%	62.4%
TypeSQL-Cont.0	**89.9%**	**92.1%**	**75.8%**
TypeSQL-Cont.1	**90.0%**	**91.9%**	**86.7%**

We run the model for different numbers of units (60, 120, 180). We observe that with 180 units the model achieves it's the highest performance



We can see the higher accuracy when the database content is available (blue line) in contrast with the simple model (content is not available). However, the cost in performance comes with bigger time consumption.



The model is tested also in the different optimizer (SGD, Adadelta, Adam) with the following results:

Units-120--Epochs-7

SGD: 12%

Adam: 61.4%

Adadelta:: 61.1%

More diagrams there are in ***Results.pdf***

Python Code

```
--------  description  --------

!python mytypesql.py: Description

!python mytypesql.py arg1 arg2 ... arg6
arg1: Folder=["Folder_For_Saving_Models"]
arg2: content=[0 | 1]
arg3: GPU=[True | False]
arg4: Optimizer=[adam|sgd|adadelta]
arg5: No units =[ 60 | 120 | 180 ]
arg6: Testing=[ yes | no ]
```

If you want to run the code, go to Code file. You can take the description of the code by running:

`python MyTypeSQL.py`

For training and Testing type;

`python MyTypeSQL.py arg1...arg6`

Training Methods provide at each iteration relative information about learning processes like Loss, TrainAccuracy, and Validation accuracy and time per epoch.

```
 * * * * * * * * * *   Epoch  7   * * * * * * * * * *
 Loss =  0.5880189823470545

TrainAccuracy 0.829807999432161
    $agg: 0.919
    $sel: 0.964
    cond: 0.931 [$cond_num, $cond_col,$cond_op,$cond_val] [0.994 0.954 0.992 0.991]


Val_Accuracy 0.7375296912114014
    $agg: 0.896
    $sel: 0.905
    cond: 0.891 [$cond_num, $cond_col,$cond_op,$cond_val] [0.987 0.935 0.981 0.988]

Epoch duration: 3206.212351 53.43687251666667
```

At the end of the training are provided all the information for all epochs.

```
* * * * * * * * * * * * *    Results    * * * * * * * * * * * * *
['MyTypeSQL.py', 'Sav_Models', '1', 'True', 'adam', '120', 'yes']

Exec time: 3085.7968290000003
Avg time: 440.8281184285715
Time= [442.182 443.744 437.246 439.498 441.637 437.203 444.287]
Exec time: 51.429947150000004
Avg time: 7.3471353071428585
Time= [7.3697     7.39573333 7.28743333 7.32496667 7.36061667 7.28
 7.40478333]

Train_Loss= [2.584 1.381 1.04  0.845 0.736 0.648 0.588]
Train_Acc= [0.402 0.604 0.712 0.762 0.797 0.808 0.83 ]
Val_Acc= [0.382 0.564 0.663 0.703 0.721 0.729 0.738]

tr_agg= [0.896 0.905 0.91  0.909 0.911 0.909 0.919]
tr_sel= [0.696 0.835 0.9   0.928 0.945 0.956 0.964]
tr_cond= [0.605 0.768 0.85  0.89  0.916 0.922 0.931]
tr_num= [0.987 0.994 0.994 0.996 0.997 0.996 0.994]
tr_col= [0.647 0.795 0.878 0.912 0.935 0.943 0.954]
tr_opr = [0.985 0.99  0.989 0.99  0.992 0.993 0.992]
tr_val= [0.985 0.989 0.988 0.992 0.992 0.991 0.991]
```

```
val_agg= [0.884 0.891 0.899 0.898 0.894 0.892 0.896]
val_sel= [0.684 0.807 0.864 0.884 0.892 0.903 0.905]
val_cond= [0.586 0.747 0.827 0.864 0.884 0.89  0.891]
val_num= [0.985 0.991 0.99  0.991 0.992 0.989 0.987]
val_col= [0.634 0.783 0.87  0.901 0.922 0.933 0.935]
val_opr = [0.983 0.987 0.983 0.983 0.984 0.981 0.981]
val_val= [0.985 0.986 0.984 0.989 0.987 0.986 0.988]

Exec time: 51.429947150000004
Avg time: 7.3471353071428585
Time= [7.3697, 7.395733333333334, 7.287433333333333, 7
```

```
 --------------  Testing  ---------------
Test acc_qm: 0.701077029665554;
  breakdown on (agg, sel, where): [0.89859545 0.89752
 0.98400202]
Test execution acc: 0.7870504503369654
Test acc_qm: 0.701077029665554;
  breakdown on (agg, sel, where): [0.89859545 0.89752
 0.98400202]
Test execution acc: 0.7870504503369654
```

You can test your model or My default model (Epochs:7, Units:180, Opt:Adam) with;

`python Testing.py [cont0 | cont1]`

```
 ----------------------------  Description  ----------------------------

    Run: python Testing.py       # Default Model (Epochs:7, Units:180, Opt:Adam )
    Run: python Testing.py Cont0 # If you have Already trained a model with Cont0
    Run: python Testing.py Cont1 # If you have Already trained a model with Cont1

Note: If you Test your Model (not Default) change the variables in this script manually
```

Sources:

WikiSQL: https://github.com/salesforce/WikiSQL

TypeSQL: https://github.com/taoyds/typesql
SQLNEt: https://github.com/xiaojunxu/SQLNet
Seq2SQL: https://github.com/shanelleroman/seq2sql