Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning

V. Zhong, C. Xiong, R. Socher

Salesforce Research

arXiv: 1709.00103

Reviewed by : Bill Zhang

University of Virginia

https://qdata.github.io/deep2Read/

Outline

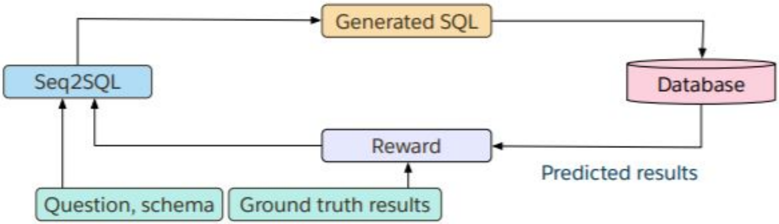
Introduction

Basic Premise and Motivation

- Relational databases are used in a vast amount of applications
- Accessing these databases requires understanding of query languages like SQL which can be difficult to master
- Thus, it can be useful to be able to translate natural language questions into SQL queries

Introduction

Summary Diagram



Model

- Task is to generate SQL query given a question and table schema
- For baseline model, use the attentional seq2seq neural semantic parser proposed by Dong Lapata (2016) which achieves state-of-the-art performance on many datasets without hand-engineered grammar
- Note that output space of baseline is unnecessarily large; we only need table columns, the question, and SQL keywords in output

Model

Augmented Pointer Network

- Generates SQL query token by token by selecting from input sequence
- Input sequence is concatenation of column names, the question, and SQL keywords
- Let x^s represent sequence of words in SQL vocab, x^q represent sequence of words in question, and x_i^c be the sequence of words in the ith column; then we can write input as

$$x = [< col >; x_1^c; x_2^c; ...; x_N^c; < sql >; x^s; < question >; x^q]$$

  where sentinel tokens separate the three types of inputs

Model

Augmented Pointer Network

- x is encoded using two-layer bidirectional LSTM; inputs to encoder are embeddings corresponding to words in input sequence
- Denote output of encoder as h^{enc}, where h_t^{enc} is state of encoder corresponding to tth word in input sequence
- Decoder uses two-layer unidirectional LSTM; at each step s, takes the previous output as input and generates state g_s
- Then, an attention score is calculated for each position of input sequence

$$\alpha_{s,t}^{ptr} = W^{ptr} \tanh\left(U^{ptr} g_s + V^{ptr} h_t\right)$$

Finally, choose input token with highest score

Model

Seq2SQL

- Note that SQL queries generally have 3-part structure: an aggregation operator followed by a SELECT column, and ending with a WHERE clause
- First two components are supervised using cross-entropy loss and third component is trained with policy gradient
- This further prunes output space of generated queries
- Overall model is trained with mixed objective function which equally weights loss function for each component

$$L = L^{agg} + L^{sel} + L^{whe}$$

Model

Seq2SQL: Aggregation Operation

- Depends on the question: for example, a question beginning with "How many..." would have a COUNT operator
- First, compute scalar attention scores for each tth token of the input sequence: $\alpha_t^{inp} = W^{inp} h_t^{enc}$
- Then, normalize to produce input encodings: $\beta^{inp} = softmax(\alpha^{inp})$
- $\kappa^{agg} = \Sigma_t \beta_t^{inp} h_t^{enc}$ is the aggregate input represenation
- Let $\alpha^{agg} = W^{agg} \tanh(V^{agg} \kappa^{agg} + b^{agg}) + c^{agg}$ represent the scores of the aggregation functions (COUNT, MIN, MAX, NULL) computed by applying a multi-layer perceptron
- Finally, use softmax and cross-entropy loss to select the operation

Model

Seq2SQL: SELECT Column

- Essentially a matching problem where we choose best column name given the question
- First, encode each column using LSTM: $h^c_{j,t} = LSTM(emb(x^c_{j,t}), h^c_{j,t-1}), e^c_j = h^c_{j,T_j}$ where j is the column, $h^c_{j,t}$ is the tth encoder state for the jth column, and last encoder state is e^c_j
- To encode the question, compute input representation κ^{sel} similarly as κ^{agg} but with untied weights
- Apply multi-layer perceptron over the column representations with $\alpha^{sel}_j = W^{sel} \tanh(V^{sel}\kappa^{sel} + V^c e^c_j)$
- Again, use softmax and cross-entropy loss to select the column

Model

Seq2SQL: WHERE Clause

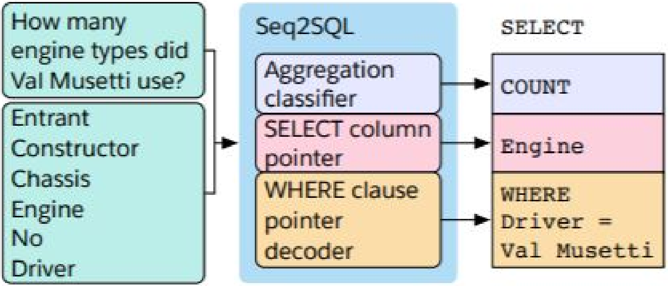
- WHERE conditions can be swapped and yield the same result, so previous methods may not work well; instead use RL policy
- Instead of teacher forcing at each step of query generation, sample from output distribution to generate next token
- Let y be the generated WHERE clause, $q(y)$ be the query generated by the model, and q_g be the ground truth query; then, define reward function

$$R(q(y), q_g) = \begin{cases} -2, \text{if not valid SQL query} \\ -1, \text{if valid SQL query with incorrect result} \\ +1, \text{if valid SQL query with correct result} \end{cases}$$

- Then, define loss from WHERE clause $L^{whe} = -\mathbb{E}_y[R(q(y), q_g)]$; derivation of policy gradient in paper

Model

Seq2SQL Graphic



WikiSQL

- Collection of questions, corresponding SQL queries, and SQL tables
- Largest hand-annotated semantic parsing dataset to date; order of magnitude larger than comparable datasets in number of tables in some cases and number of examples
- WikiSQL has many table schemas and has realistic data from the web
- Collected by crowd-sourcing on Amazon Mechanical Turk in two phases: (1) Worker paraphrases generated question for table, where questions are made based on a template, (2) 2 other workers verify that paraphrase matches question, also discarding ones without enough variation
- Dataset randomly divided into training, dev, and test sets

WikiSQL

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

WikiSQL

Evaluation

- Let N be the number of examples in the dataset, N_{ex} be the number of queries that have the correct execution result, and N_{lf} be the number of queries which exactly match the ground truth query
- Use accuracy metrics $Acc_{ex} = \frac{N_{ex}}{N}$ and $Acc_{lf} = \frac{N_{lf}}{N}$
- Cannot only use ex accuracy because sometimes multiple queries give the same result even if they aren't the same query
- Cannot only use lf accuracy because sometimes the WHERE clauses switches the order clauses, thus not exactly matching ground truth

Experiments

- Tokenize dataset using Stanford CoreNLP, using normalized tokens for training and converting back into original gloss before outputting query
- Used GloVe word embeddings and character n-gram embeddings
- All networks run for at least 300 epochs with early stopping
- Traing using Adam optimizer and dropout
- All recurrent layers have hidden size 200 and 0.3 dropout
- WHERE clause training is supervised using teacher forcing (not trained from scratch) and continued with RL

Experiments

Result

- Seq2SQL achieves state-of-the-art; to make baseline more competitive, augment the baseline's input too

Model	Dev Acc_{lf}	Dev Acc_{ex}	Test Acc_{lf}	Test Acc_{ex}
Baseline (Dong & Lapata, 2016)	23.3%	37.0%	23.4%	35.9%
Aug Ptr Network	44.1%	53.8%	43.3%	53.3%
Seq2SQL (no RL)	48.2%	58.1%	47.4%	57.1%
Seq2SQL	**49.5%**	**60.8%**	**48.3%**	**59.4%**

Experiments

Analysis

- Limiting the output space via pointer network leads to more accurate conditions
- Incorporating structure reduces invalid queries
- RL trains higher quality WHERE clauses which can be ordered differently from ground truth

Related Work

Semantic Parsing

- Natural language questions are parsed into logical forms that are then executed on a knowledge graph
- Typically constrained to single table schema and require hand-curated grammars to perform well
- Some modifications allow for generalization to new table schema, but Seq2SQL does not require access to table content, coversion of tables to graphs, or hand-curated grammars
- Many datasets are single-schema and overall limited in scope

Related Work

Representation Learning for Sequence Generation

- Dong Lapata (2016) has state-of-the-art seq2seq neural semantic parser without need for hand-engineered grammar
- Vinyals et al. (2015) has similar pointer-based generation
- Many more in paper

Related Work

Semantic Parsing

- PRECISE (Popescu et al., 2003) translates questions to SQL while identifying questions it is unsure about
- Giordani Moschitti (2012) generates candidate queries from a grammar and ranks them using tree kernels
- Above two require high quality grammars and do not generalize to new schema
- Iyer et al. (2017) uses a seq2seq model improved by human feedback
- Seq2SQL's RL training outperforms seq2seq

Conclusion

- Seq2SQL leverages structure of SQL to reduce output space of model
- Applied in-the-loop query execution to train Seq2SQL since cross-entropy loss is unsuitable
- Introduced WikiSQL, a dataset of questions and SQL queries an order of magnitude larger than comparable datasets
- Showed that Seq2SQL outperforms state-of-the-art semantic parsers on WikiSQL

References

- https://arxiv.org/pdf/1709.00103.pdf