Natural Language Query Translation To Cypher Query

Akshay Shinde
Oregon State University
Corvallis, OR
shindea@oregonstate.edu

Rahul Damineni
Oregon State University
Corvallis, OR
daminens@oregonstate.edu

Sheekha Jariwala
Oregon State University
Corvallis, OR
jariwals@oregonstate.edu

Abstract

In this paper, we present the approaches followed to translate Natural language questions to Cypher query - query language for graph database called Neo4j[2]. We have used the dataset WikiSQL introduced in the paper by Zhong et al [13]. We draw equivalence between basic data-structures between Relational database (that is, table) and Graph database (that is, graph) to guide the translation process. We studied the approaches followed to convert Natural Language query to SQL query and extend the model to generate Cypher query. In the end, we also release first-of-a-kind dataset called WikiCypher, collection of natural language questions, nodes and Cypher queries.

1. Introduction

Traditionally, relational databases have been used to store data by myriad of enterprises in form of structured tables defined by columns of specific type and rows of instances of data. However, relational databases are not very suitable to store the relationships between entities represented by tables. As the trend of storing information about entities and relationships between them increases, graph databases become a natural choice to store data of this nature. Graph databases provide a natural way to model entities and relationships by using nodes, edges and properties. There are multiple vendors of graph databases like *Neo4j, ArangoDB, OrientDB, etc*. According to a leading source [3] *Neo4j* has been a highly popular graph database since 2017. *Cypher* is a query language used to query *Neo4j* graph database.

Semantic parsing is a popular task in the field of Natural Language Processing which deals with converting human-understandable natural language to a machine-understandable logical form allowing it to further process it for numerous downstream tasks. A very widely researched semantic parsing task is NL2SQL (Natural Language to SQL). Our work deals with a variant of NL2SQL task viz. converting natural language questions to queries in *Cypher*.

Owing to long-standing usage of relational databases, semantic parsing problem has been researched extensively to convert natural language queries to SQL, which is the query language for relational databases. However, as graph databases start to gain more popularity, the field of semantic parsing to translate natural language queries to Cypher is ripe for research. We derive lessons from research in NL2SQL task to tackle the problem of NL2Cypher. Specifically, we look at the *WikiSQL* task [13] which introduced a dataset containing 80654 instances of natural language queries, SQL queries and corresponding 24241 tables from Wikipedia. It's the largest dataset among other known datasets known for semantic parsing task.

To work on NL2Cypher task, there was just one known dataset CLEVR [1] which contains set of graph, question and answer tuples. However, data in this dataset is modelled on a narrow domain of transit networks. Corresponding questions are modelled around these transit networks limiting the scope learned by model and ability to translate natural language queries from unobserved domains. On the other hand, there are numerous Neo4j based graph datasets without corresponding natural language questions and answers. Thus, we chose to work with *WikiSQL* dataset which covers wide number of domains from Wikipedia and also has sizable number of question-answer pairs.

To develop a general-domain model which translates natural language queries to Cypher, we studied related work to translate natural language to SQL which is also a structured language. In the next section, we review the related work which deals specifically with WikiSQL task.

2. Related Work

[13] (Zhong et al. 2017) introduced *WikiSQL* and proposed a vanilla sequence-to-sequence model which acted as baseline. Authors quickly proposed a deep neural attentional sequence-to-sequence model called Seq2SQL which uses reinforcement learning to guide learning of two components - *select* clause and *where* clause. Their model has three components which leverages structure of SQL query to limit the output space of predicted queries. Their model

also uses policy-based reinforcement learning (RL) technique to generate the conditions of the query, which cannot be optimized well using cross entropy loss due to their unordered nature.

Another model SQLNet proposed by (Xu et al. 2017) [10] achieved a better accuracy of translating to SQL query without using reinforcement learning. *SQLNet* avoided using sequence-to-sequence in decoder of the model because the order of clauses in *where* clause doesn't matter. Instead it employs a sketch-based approach, where the sketch aligns naturally to the syntactical structure of SQL query. That is, SQLNet, is used to predict the content for each slot in a sketch or a template which essentially containing a dependency graph. Authors propose two approaches to predict each slot based on predictions of other slots that it depends on - *sequence-to-set* and *column attention*. *sequence-to-set* predicts an unordered set of constraints and *column-attention* which captures the dependency relation between slots defined in sketches during inference phase. The six modules being predicted are *aggregation function*, column corresponding to *select* clause, *where* clause, *operator* and value corresponding it.

(Wang et al., 2018)[9] proposed technique called *execution-guided decoding* which uses the idea that partially generated SQL queries can be executed and the generated results can be used to direct the generation of rest of the SQL query. All of these ideas are culminated into the model *SQLova* by *Hwang et al* [8] which uses table-aware word-embeddings from BERT [7] model as encoder. The same model proposed a simple *shallow-layer* architecture on top of the BERT-based encoder. Finally, the generated candidate SQL queries are filtered by using *execution-guided decoding* proposed by Wang et al [9].

3. Our Approaches

To work with the WikiSQL dataset, we need to understand the equivalence between Relational database and Graph database. We show this equivalence by using the following table:

Relational database	Graph database
Table and relationships by using joins	Nodes and Edges
Table	Node type
Columns	Node properties
Rows	Node instances

Keeping this equivalence in mind and learning from the approaches to translate natural language to SQL, We propose several approaches to translate natural language queries to Cypher queries as explained in following subsections:

3.1. SEQ2Cypher

For NL2SQL models, input is a vector containing tokens of natural language query and column headers. Equivalently, our SEQ2Cypher baseline model is a sequence-to-sequence model which takes natural language query and node properties as input to generate sequence of Cypher query.

```
[SOS][..NLQ..][SEP][...table_name...][
    SEP][...headers...][EOS]
```

We train a WordPiece tokenizer by using Huggingface's tokenizers library [4] on train split of the dataset to tokenize the input which is passed through Embedding layers and GRU units. Decoder consists of Embedding layers and GRU units to decode the context vector. It's important to note that the output is decoded by using another WordPiece tokenizer which has been trained on sample Cypher queries.

```
[SOS][...cypher_query...][EOS]
```

The model uses LogSoftmax loss function to compute the loss.

3.2. NL2SQL2Cypher

Another alternative to perform NL2Cypher is to use SQL queries generated by any of the existing models which generate SQL queries from natural language queries and convert the SQL queries to Cypher queries by using a sequence-to-sequence model. We do this by two methods:

- Sequence-to-Sequence based approach - This model takes SQL query, node type and node properties as input.

- OpenNMT approach [5] - We used OpenNMT library to train a vanilla 2 layer LSTM-based model to predict Cypher queries from input SQL queries with

  ```
  [SOS][..SQL..][SEP][...table_name
      ...][SEP][...headers...][EOS]
  ```

  We train a WordPiece tokenizer by using Huggingface's tokenizers library [4] on train split of the dataset to tokenize the input which is passed through Embedding layers and GRU units. Decoder consists of Embedding layers and GRU units to decode the context vector. It's important to note that the output is decoded by using another WordPiece tokenizer which has been trained on sample Cypher queries.

  ```
  [SOS][...cypher_query...][EOS]
  ```

  This model also uses LogSoftmax loss function to compute the loss.

4. Experiments

We took two approaches to generate Cypher queries. One is directly generating from natural language query and the another one is to generate from a SQL query. In each approach, we trained two models: both sequence to sequence, GRU based encoder-decoder models but one of these uses attention on encoded input sequence while decoding. This means, we've conducted 4 experiments. Each model is fine-tuned and found to be optimally performing with these hyper parameters:

Architecture related:

- Number of GRU layers: 1
- Number of hidden units in encoder/ decoder: 56
- Activation function: LeakyReLU
- Embedding trained from scratch

Regularization:

- Dropout ratio for attention 0.9
- Encoder hidden state dropout 0.9
- Optimizer with weight decay torch.optim.AdamW defaults

Evaluation: We minimised Negative Log Likelihood Loss and aimed to maximise logical form accuracy.

Results: For each experiment, we split data into training and testing portions and logged normalized loss and normalized logical form accuracy. Due to time constraints and technical difficulties related to GPU servers, we were able to train our deep neural network only on 1% of our original data on CPU. 1% of our data is about 700 examples. So the results didn't turnout quite remarkable.

Experiment Name	Train Accu	Test Accu	Train Loss	Test Loss
NL2Cypher (encoder-decoder)	0.72	0.24	1.84	7.21
NL2Cypher (attention based)	0.72	0.23	1.85	6.70
SQL2Cypher (encoder-decoder)	0.72	0.20	1.88	7.48
SQL2Cypher (attention based)	0.72	0.27	1.85	6.83

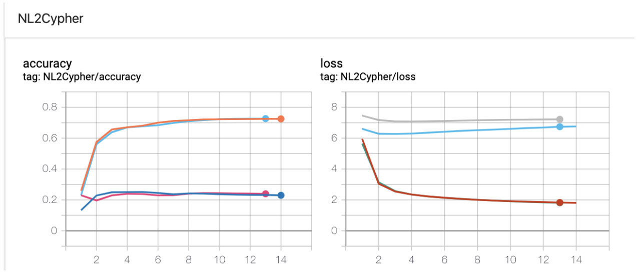


Figure 1. Natural Language to Cypher Queries

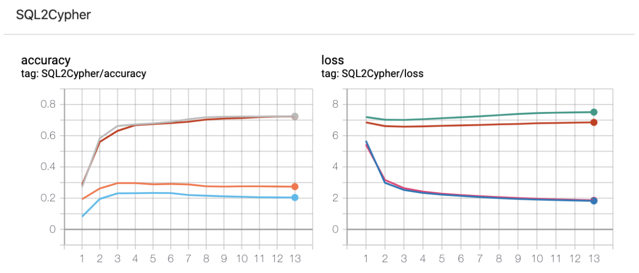


Figure 2. SQL to Cypher Queries

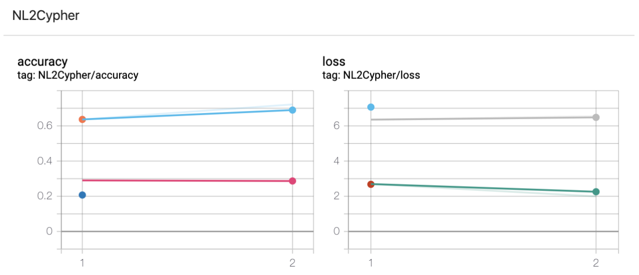


Figure 3. Natural Language to Cypher Queries, model trained on 10% of original dataset

Training the same models on 10% of our data just for two epochs already overtook the best accuracy set by 1% data after 11 epochs. This indicates that our training procedure and model complexity is adept and if trained on full data set, this could achieve respectable translation accuracy

5. Failed Experiments

- We tried using BERT as base and made slot based modules to set it up as a classification task as described in SQLova paper but fine-tuning BERT was highly resource consuming.
- We used OpenNMT to train SQL2Cypher model using out-of-box scripts but the performance after several iterations was poor.
- Other attempts are listed as branches in our GitHub repo `https://github.com/`

ednihs-yahska/CS535_final_project/
branches

6. Evaluation

The models built to translate natural language query to structured language are usually evaluated by using two types of accuracies as explained below:

- *Logical form accuracy* - This is derived from string by string comparison of predicted query and actual query. Mathematically, this accuracy is derived by dividing the number of queries which are correctly predicted token-by-token by total number of queries in the data split. It's crucial to note here that this is not the most precise evaluation metric because a SQL query can be written in multiple ways to fetch the same data from table.

- *Execution accuracy* - This is extended form of accuracy which gives a better understanding of query prediction than logical form accuracy. This accuracy is derived by dividing the number of predicted SQL queries which when executed gave the same results as ground-truth SQL query.

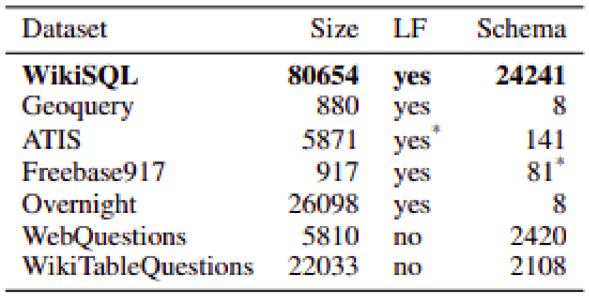
For our models, we're reporting logical form accuracy only.

7. WikiCypher

There has been extensive research done in area of semantic parsing of natural language queries being translated to structured language like SQL. However, there was no such general-domain dataset which was huge enough to build a deep-learning model. And as seen from Fig 4 WikiSQL[13] was the biggest general-domain semantic-parsing dataset.

A unique characteristic of this dataset is that it contains real-life data extracted from Wikipedia with varied Natural language questions have varied lengths, corresponding SQL questions and tables have a distribution as seen by Figure 5 6 7 and 8

In view of these characteristics, we introduce WikiCypher - dataset derived from WikiSQL with Cypher queries and graph data. WikiCypher is collection of natural language queries, corresponding Cypher queries and node types, properties with node instances. This dataset has been prepared heuristically from WikiSQL dataset giving 80651 instances with 26528 nodes types. Number of examples and corresponding number of node types for each of the data splits is listed in following table.



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 4. Comparison of WikiSQL with other semantic parsing datasets. Here, LF indicates whether the dataset has logical-form annotation, that is, whether the questions and corresponding tables have corresponding SQL queries. Size indicates the number of examples in the dataset and Schema indicates number of corresponding SQL tables.

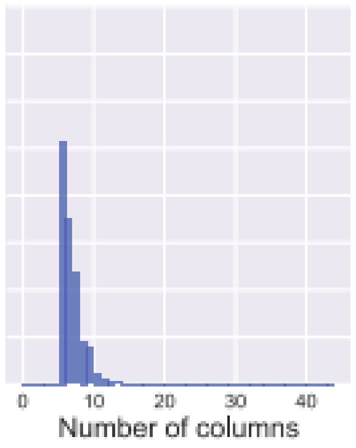


Figure 5. Distribution of number of node properties

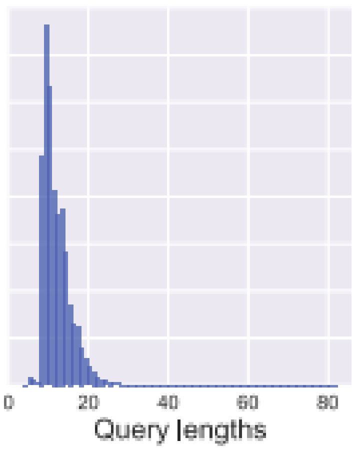


Figure 6. Distribution of Query Lengths

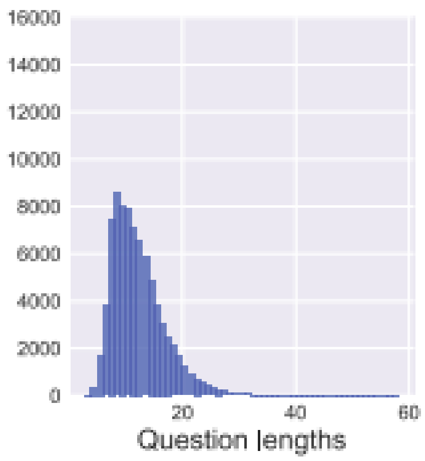


Figure 7. Distribution of Natural Language Question Lengths

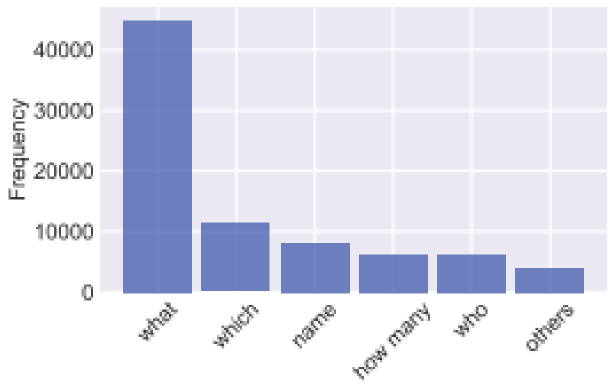


Figure 8. Distribution of domains of Natural Language Questions

	train	dev	test
NL questions	56354	8420	56354
Node types	18584	2715	5229

Number of natural language examples and number of node types in train, dev and test splits of WikiCypher dataset

The dataset can be downloaded from here[6] https://github.com/ednihs-yahska/WikiCypher

8. Conclusion

We built the first deep-learning model to translate natural language queries from WikiSQL dataset to Cypher queries. We experiment with a deep-neural sequence-to-sequence architecture which gives logical form accuracy of %. We also added attention to sequence-to-sequence model to see improved results. Another approach was a sequence-to-sequence architecture to convert SQL queries generated from SQLova model [8] to Cypher queries by using OpenNMT library.

Lastly, we also release WikiCypher dataset, collection of natural language questions, Cypher queries and node types for future work in this domain.

All our work is open sourced at `https://github.com/ednihs-yahska/CS535_final_project`

9. Future Scope of Work

Our *WikiCypher* dataset can be used to develop deep-learning models for semantic parsing tasks to generate Cypher queries. By using similar heuristics, *Spider* dataset which is a cross-domain semantic-parsing dataset from (Yu et al. 2018) [12] can be translated to Cypher queries as well.

Our baseline models (SEQ2SQL and SEQ2SQL without attention) can be extended to use pretrained models like BERT to generate embeddings as done in SQLova model by (Hwang et al., 2019) [8]. We propose usage of XLNet model by (Yang et al., 2019) [11] which is found to have better than accuracy than BERT on 20 tasks. We reckon that XLNet will improve accuracy of NL2Cypher task as well.

References

[1] Clevr graph: Dataset for graph question answering.

[2] Cypher - query language for graph database.

[3] db-engines.com - trends of graph database popularity.

[4] Huggingface tokenizers - fast state-of-the-art tokenizers optimized for research and production.

[5] Opennmt - open source neural machine translation in pytorch.

[6] Wikicypher - collection of natural language queries, cypher queries and corresponding node types.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *NAACL*, abs/1810.04805, 2018.

[8] W. Hwang, J. Yim, S. Park, and M. Seo. A comprehensive exploration on wikisql with table-aware word contextualization, 2019.

[9] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh. Robust text-to-sql generation with execution-guided decoding, 2018.

[10] X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017.

[11] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019.

[12] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, 2018.

[13] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.