Developing a Natural Language Interface to Databases using character level embeddings

Sarim Zafar
dept. of Computer Science
North Dakota State University
Fargo, USA
sarim.zafar@ndsu.edu

***Abstract*—Much of the data in the world is stored in databases and to access that data users are required to have command over SQL or equivalent interface language. Hence, using a system that can convert a natural language utterance to equivalent SQL query would make the data more accessible. Using natural language for Synthesizing SQL queries, which falls under the umbrella of semantic parsing, is a widely studied field and recently found momentum again due to the introduction of large-scale datasets. These datasets usually contain a natural language utterance that maps to a SQL query. We would also need additional information alongside the utterance to make an appropriate prediction as we cannot expect the user to state every aspect of the query in their statement. Therefore, usually schema is also provided to aid the prediction process. Various neural network based approaches have been proposed over the past few years addressing this problem. The solution proposed in these studies usually involves using word level embeddings such as GloVe alongside LSTM. In this paper we show that by using Character level embeddings with Gated Recurrent Units, we can get comparable results.**

***Index Terms*—Databases, semantic-parsing, Nueral Networks**

I. Introduction

The field of semantic parsing for databases has been studied for quite a long time, but only recently it has been reinvigorated by the introduction of large-scale datasets that enable training neural network-based approaches to solve the problem. Before the introduction of such massive datasets, we had small datasets [1]–[6] which had only a few hundred to a couple of thousand examples. This forced researchers to come up with solutions that were either too specialized to one particular kind of database or had to be actively be modified to accommodate to address corner cases. Although there are multiple large-scale datasets available such as NLIDB [7], SPIDER [8] and WIkiSQL [9], for the purpose of this project we will work with WikiSQL only For the scope of this project.

Our goal in this project is nor to produce state of the art nor to introduce any breakthrough algorithm. Instead, it is to evaluate some existing options that were often overlooked to solve this problem and see if they can offer some merit for further consideration in the future. Pretty much all of the recently developed systems use a modular approach and ours is no different, though on a smaller scale (fewer parameters) due to hardware limitations. Our contribution is to use a Siamese network [10] based approach for two of the modules out of five and utilizing character level embeddings for all of the modules. We also replace the traditionally used LSTM variant of recurrent neural networks with GRU.

```
SELECT AGG(col)
  FROM TABLE
   WHERE (COL OP COND)
  [AND WHERE (COL OP COND)*];
```

Fig. 1. Query structure

II. Dataset

The WikiSQL dataset contains 80,654 pairs of natural language questions and their respective SQL queries. There are 24,241 tables which originate from Wikipedia. Each question is only related to only one table; hence the structure of the output queries is not as complicated as it could be in the case for multiple tables. By only requiring the schema and a natural language question for query generation we also avoid the scalability and privacy issues that we might have faced in other scenarios. The structure of the SQL queries can be summarized as shown in fig 1.

So, as you can see, we are left with a relatively simple structure. Because the dataset was hand-annotated this helps to avoid overfitting that might have been caused by template synthesized descriptions. Lastly the train dev and test set are all disjoint which means that they don't share the schema which enables us to evaluate the generalization of our approach. In fig 2 we can see the distribution of different types of question asked. In fig 4 we can see the histogram of lengths of questions/SQL queries the dataset along with the distribution of the number of columns in each table.

III. Related Work

As briefly mentioned in the introduction, we will deal with WikiSQL dataset; hence, it is only appropriate to discuss previous research work that have evaluated their approach on the mentioned dataset. Most of the approaches use a sketch syntax and use the dependency in the sketch to perform slot filling via modules. For the scope of this report, we will only discuss three approaches.

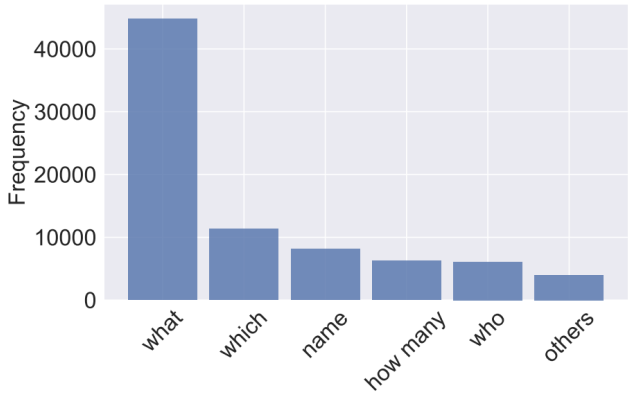

Fig. 2. Question type distribution

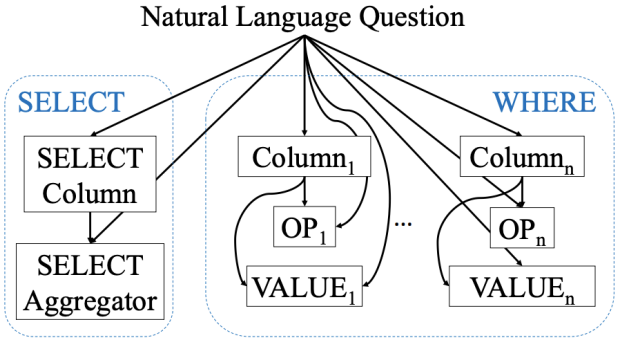

Fig. 3. Dependency Graph

Zhong et al. [9] purposed a modular approach that took advantage of the simple structure of the SQL queries in the dataset. They also purposed a baseline seq2seq model to draw a comparison. The baseline seq2seq model used pointer networks. Pointer networks were introduced by vanyals et al. [11] to solve seq2seq problems.

The Input of the model was tokens, including the columns of the table, SQL grammar and the user query. The output was the whole SQL query. They used Bidirectional LSTM to encode the sequence then the encoding was fed to a pointer LSTM to decode. The Pointer network chooses from the tokens fed as the input sequence to create the query. The modular approach consisted of three modules AGG, SELECT, and WHERE. They used pre-trained GloVe embeddings in their modules to encode the text [12]. SELECT and AGG modules were simple classification modules in which they classified which column for SELECT part of the query is required. Then the AGG module is used to determine if an aggregation over the said column is required and if so then which one. However, the WHERE module is a bit more complicated. It was trained as a simple seq2seq problem. The order of where clause determined whether the generated query was correct or not if cross-entropy loss was employed. But because they are joined together with AND the order does not matter. This problem cannot be addressed by only using a cross-entropy loss for a seq2seq module especially in this setting. So, the authors purposed a Reinforcement Learning based approach where the model would query the database using the generated WHERE clauses and penalize based on success or failure.

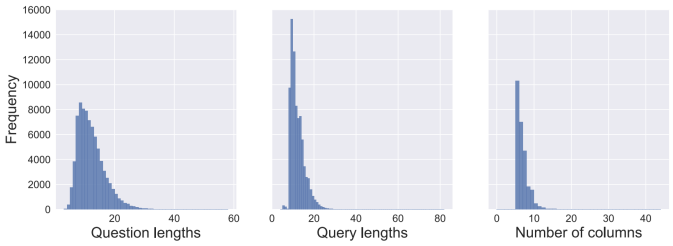

Fig. 4. Distribution of table, question, query sizes in WikiSQL

Xu et al. [13] reasoned that the problem is rather Seq2Set rather than Seq2Seq. They also showed that solving the problem through RL is wrong as multiple queries that are not even logically correct can have the same results hence breaking down the WHERE module into three further modules WHERE-COLUMN, WHERE-OP and WHERE-Value. They only used pointer network for the 3rd module as the rest of them could be cast as a classification problem. Their approach proposed the dependency graph shown in fig 3, which shows the dependency graph and the modules of the proposed pipeline. These changes allowed them to improve over Zhong et al. [9] by around 9%.

Hwang et al. [14] purposed that rather than using GloVe embeddings we can take advantage of recent advancements in NLP and use BERT embeddings [15] instead. They also purposed that rather than predicting the whole VALUE slot through a pointer network, we should instead predict the start and end index from the users query. They are SOTA at the time of writing, with their test score being almost 90% for WikiSQL task.

IV. Approach

Our key insight for choosing character level embedding instead of word level is two folded. One, we believe that this brings down the complexity a lot. The difference between the number of unique characters vs. unique words is astronomical. Secondly, even with a single typo, we would have to create a new token index for that word, but with character level embeddings we would not have to accommodate this but rather help the model understand that these slight deviations are just noise. Our other critical insight is the usage of GRU instead of LSTM that is traditionally used in previous studies. In many NLP related works, it has been shown that GRU can yield similar performance of LSTM while being computationally more efficient. In this work, we evaluate this hypothesis as well. Our purposed pipeline can be broken down into six modules. We also follow the slot filling approach as other approaches.

A. SELECT Module

The main purpose of this module is to find the appropriate SELECT column given the natural language utterance. Traditionally this is treated as a classification problem. Func(Col(i)|Q)=T/F, where i is the ith column of the table and Q is the question. These tokens are converted into an embedding then that is passed through a bidirectional LSTM, and lastly a sigmoid function is applied to give a score between 0 and 1. We propose using siamese network approach. After passing these sequences through character level embeddings and then passing them through a bidirectional GRU, we extract the hidden states of the GRU. Then we calculate the euclidean distance between these two representations and use it as a loss function to train the network. So what we are trying to do in essence is to minimize the distance between the learned representation of these two input sequences. We choose to say that the two representations are the same if the euclidean distance between them is less than 0.5. Fig 5 shows the visualization of the particular module. We tried experimenting with batch normalization as it is traditionally used as a remedy if the the network is not converging fast enough because it zero means the data and ensures one std along the given axis which in our case was the last one. However, we did not notice any advantage in our case for this module, particularly.

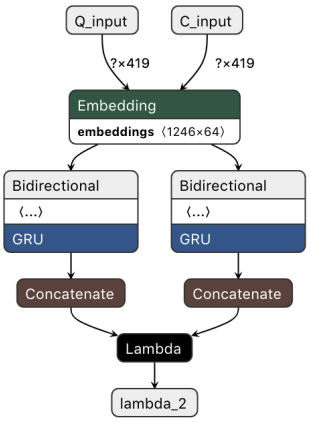

Fig. 5. SELECT Module computation graph

B. AGG Module

The main purpose of this module is to find the appropriate AGG operation that needs to be performed given the natural language utterance and the selected column in the SELECT clause. This can be posed as a classification over six classes as following [”, ’MAX’, ’MIN’, ’COUNT’, ’SUM’, ’AVG’]. So we select one of the six classes conditioned on the selected column yielded from the Select module and user query. After passing these sequences through character level embeddings and then passing them through a bidirectional GRU, we extract the hidden states of the GRU. Then after concatenating the representation, we pass it to a dense layer which has softmax function which yields a probability distribution over all the classes and we choose the one with maximum probability. Fig 6 shows the visualization of the particular module.

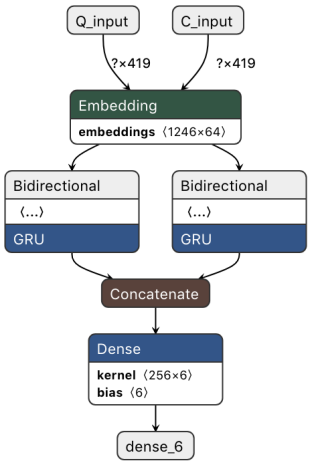

Fig. 6. AGG Module computation graph

C. WHERE K Module

The primary purpose of this module is to determine the number of where clauses that are needed to create the target query. Therefore, the model gets a sequence of tokens representing the natural language utterance and gets its character level embedding. The embedding is then sent into a Bidirectional GRU whose internal states are concatenated and passed to a dense layer which then applies a sigmoid function. This can be posed as a classification as we have five classes as following [0,1,2,3,4]. Detailed visualization of the particular module can be observed in Fig 7.

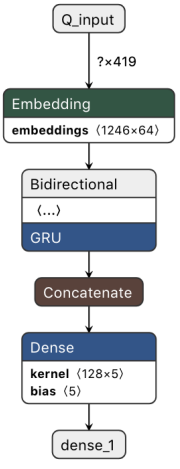

Fig. 7. WHERE K Module computation graph

D. WHERE Col Module

The task of this module is similar to SELECT module, but now rather than finding one particular column for the SELECT clause we now want the module to generate similar

representations between each of the columns that will be utilized in the SQL query and the natural language utterance. We again used a Siamese network, but the only difference was we added a batch normalization layer in this particular module as it helped it converge faster. One interesting thing we noticed while training this network was that there seems to be a ceiling that we quickly reach within a few epochs and no matter how we tried to top that we couldnt. We think it would be interesting to see if it was caused by some bug in the code from our side or is there something interesting going on in the data or if it is just some inherent limitation of our puny network. Detailed visualization of the particular module can be observed in Fig 8.

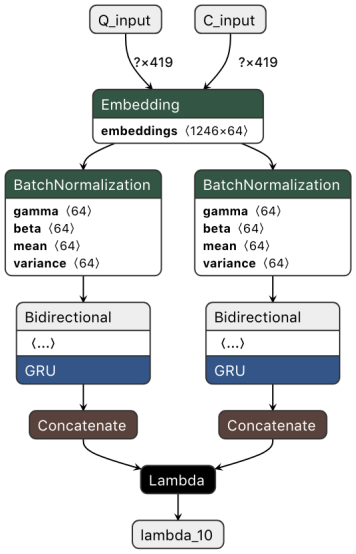

Fig. 8. WHERE COL Module computation graph

E. WHERE Ops module

This module is also a classification module with the following classes $[=, >, <]$. Given the selected column for where clause and the natural language utterance, we predict which Operation should be chosen for this particular where clause. Detailed visualization of the module can be observed in Fig 9.

F. WHERE Value module

This modules task is to predict the starting and ending index of the piece of string that needs to be used in the where clause's value slot, as shown in Fig 3. This module was a bit more complicated than the others as can be seen in Fig 10. We also include the selected Operator for the particular where clause along with the user query as a prior. We then concatenate the hidden state representations along with Ops priors and use that to predict the starting index of the sequence by passing it to a dense layer. Then we use the prediction of that particular dense layer is also concatenated with the input and passed to another dense layer that is responsible for predicting the ending index. We add dropout layer to avoid overfitting.

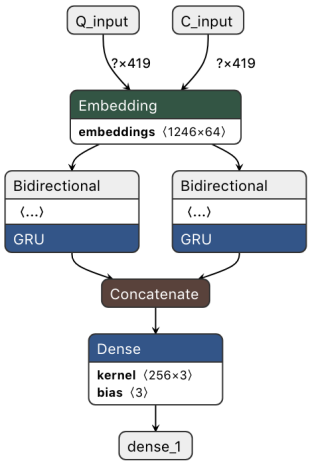

Fig. 9. WHERE OPS Module computation graph

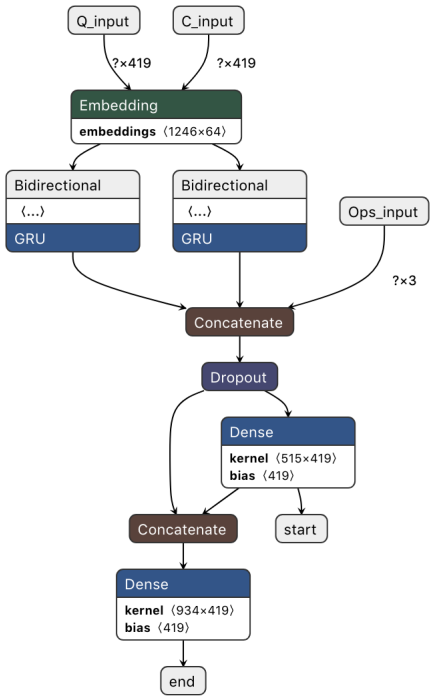

Fig. 10. WHERE VALUE Module computation graph

V. Technical Details

In this section, we will briefly go over some of the parameters involving the neural networks. Such as the embeddings were not shared across modules. All the embeddings and GRU layers had 64 dimensions/units. To train the networks, we utilize ADAM optimizer with a learning rate of 0.001. We train each of the modules up to 100 epochs and choose the network weights that have the lowest validation loss for each

Approach	AGG	SELECT	WHERE-COL	OPS	WHERE-K	WHERE-VALUE
Ours	89.1/89.2	97.06/96.64	80.06/79.29	93.00/93.20	91.84/91.68	57.92/52.97
SQLova	90.5/ 90.6	97.3/ 96.8	94.7/ 94.3	97.5/ 97.3	98.7/98.5	95.9/95.4

TABLE I
ACCURACY OF MODULES OVER DEV/TEST SET

of the task. For some of the tasks such as SELECT module, we under-sampled the majority class as it was making it difficult to train an unbiased network. We tried adding class weights to offset the problem but it was not as helpful as under sampling.

VI. Results And Conclusion

As you can see from the table above our proposed methodology yields a close score to the SOTA except for Where Col and Where Value modules. We believe its because of size of our model. Also, they used pointer networks with attention while we were just utilizing a traditional bidirectional GRU. As for the Siamese network, in hindsight, we believe that even though it is an alternative approach, but the traditional method might be a better way to address the problem. As for future, we believe that utilizing both embeddings might be more beneficial rather than just using one. As well as evaluating CNN for this task might be interesting as we know that they are computationally more efficient than Recurrent Neural Networks and can often perform as well as them on NLP tasks.

Acknowledgment

I would like to thank Dr. Anne Denton and Dr. Zubair Malik for their continued support throughout this project. I would also like to thank Pun Ajjimaporn for the healthy discussion that helped me look at this problem in a different light. Figure 2 and 4 are taken from the original Seq2SQL paper while Figure 3 is taken from SQLNET paper.

References

[1] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The atis spoken language systems pilot corpus," in *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

[2] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the national conference on artificial intelligence*, 1996, pp. 1050–1055.

[3] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates, "Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 141.

[4] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a neural semantic parser from user feedback," *arXiv preprint arXiv:1704.08760*, 2017.

[5] F. Li and H. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73–84, 2014.

[6] N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig, "Type-and content-driven synthesis of sql queries from natural language," *arXiv preprint arXiv:1702.01168*, 2017.

[7] F. Brad, R. Iacob, I. Hosu, and T. Rebedea, "Dataset for a neural natural language interface for databases (nnlidb)," *arXiv preprint arXiv:1707.03172*, 2017.

[8] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.

[9] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[10] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," in *Advances in neural information processing systems*, 1994, pp. 737–744.

[11] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[12] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[13] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," *arXiv preprint arXiv:1711.04436*, 2017.

[14] W. Hwang, J. Yim, S. Park, and M. Seo, "Achieving 90% accuracy in wikisql," 2019.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.