Search Like a Human: Neural Machine Translation for Database Search

Geoffrey B. Boullanger
Stanford University
gbakker@stanford.edu

Maxime Dumonal
Stanford University
mdumonal@stanford.edu

Abstract

With an ever growing amount of data generated every day, the need to simplify the treatment of those data becomes a requirement. We need to develop new tools to allow non-technical users to access and filter data as easily and close to the human language as possible. In this paper, we implement the state-of-the-art model for generating an SQL query from a natural language question. The model consists of three parts: a contextualized word model representation (e.g. ELMo, BERT, etc.); a sequence-to-SQL generation decoder; and an Execution-Guided (EG) decoder to guarantee the consistency of the SQL request. We chose to experiment with a different contextualized word model representation, the recently published GPT-2 model, to test the robustness of the SOTA core model; and change additional features of the model (using Transformers). We observe that this approach, along with NL2SQL suggested by SQLova [11], generates some very promising results and scores the third place on the Salesforce WikiSQL leaderboard. Our implementation with OpenAI GPT-2 gets a 77.5% logical form accuracy and 84.4% execution accuracy on the test set (compared to 83.6% and 89.6% for SQLova, respectively). Our findings prove GPT-2 (small) to be a solid multitasking pre-model, but yet doesn't produce results as good as BERT (small).

1 Introduction

The amount of data generated and collected by the world continues to grow exponentially each day, with a large percentage stored in relational databases. The current method for quickly accessing this data requires knowledge of the syntax, table schema, etc. of the query language. However, finding a general reliable method for translating human natural language questions into valid database queries has widespread applications beyond SQL. A general methodology could be applied to other database types, such as MongoDb, T-SQL, ElasticSearch to name a few.

The primary goal of our project is to implement a state-of-the-art complex Neural Machine Translation model. In this case, the translation is from human natural language (English) to Structured Query Language (SQL). The secondary goal is to push the current limit of translating accurate structural database queries by improving this model.

We have selected the current top performing algorithm[1], the SQLova model, to reimplement and find ways to improve. The Salesforce WikiSQL Competition allows us to work within a framework with multiple starting advantages: more than a dozen diverse models have been proposed with good documentation; a standardized WikiSQL dataset[2] already divided into training, development and test subsets for like-for-like comparison; and rapidly improving performance metrics.

[1]Best performing algorithm for the Salesforce WikiSQL 32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada competition for generating SQL queries from human utterances using a large crowd-sourced dataset.

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

The motivation for our project is to create a simple, fast tool for database search from human utterances without the need for specialized database knowledge. Such a tool could have broad applicability and wide usage.

Our approach can be divided into three different steps: reproduce the SOTA model; test the robustness by changing the language model used as contextual word representation; and finally change the inner structure of the weakest module of the SQLova model.

We have created a live (web) demo of the model which can be run against any SQL table. Screenshots are available in the appendix 4 and a link can be shared upon request.

As we will see in the rest of the paper, our key contribution is the use of a newer contextual embedding multitask model with a semantic parsing model for SQL, and the proof of the robustness of such a general architecture.

2 Related Work

Since WikiSQL dataset was introduced in the late 2017 by the Salesforce team, many improvements have been made to the initial model [5]. An important step was the SQLNet [4] which further simplifies the generation task by introducing a sequence-to-sql model in which only where condition value is generated by the sequence-to-sequence model. Coarse-2-Fine [7] used a 2 step encoding-decoding structure with a coarse intermediate output. The complex model presented in this paper is the latest in a series of models that have led to rapid improvement in performance, taking advantage of the development of multitask language models, here BERT, based on the Transformer model [3]. The paper has been submitted for publication in a peer-reviewed conference proceedings and is currently under review. In the beginning of March, a team at Microsoft has released a new SOTA for this task [10], increasing the accuracy by 2.0% compared to the former leader, SQLova. In their paper, the team made use of the new Multi-Task Deep Neural Networks for Natural Language Understanding (MT-DNN) model [13] released by Microsoft a month before that, where they claimed better results compared to BERT.

3 Approach

3.1 Our model genesis

To implement our model, we first adopted the overall experimental approach of first re-implementing the SQLova model [3], then modifying this baseline to try to improve performance. This required rewriting large parts of the SQLova starting code to make it run successfully and replicate the originally published training, dev and test accuracy (with Execution Guided, or EG), as reported by the SQLova team [12]. Additionally, it allows us to more easily make model changes for our proposed improvements.

Another important part of the refactoring effort was to enable us to use the model on multi-gpus and add tensorboardX[4] to the code. Adding TensorBoard[5] to the codebase helped us follow the performance of different metrics - e.g. accuracy levels for each module of the nodel. Unfortunately, despite our efforts, running on multiple GPUs was limited by the structure of the model - i.e. mostly RNN/LSTM - which is one of the reasons why we wanted to move from LSTMs to Transformers.

Keeping in mind that a lot of the model performance was due to the use of BERT layer upstream of a more dedicated task structure for the SQL generation (called `pre-model` in our code), we decided to try an even more recent language model proposed by OpenAI, GPT-2[6], which was made available on February 14th this year. However, OpenAI team took a controversial decision by not releasing their biggest (and most powerful) model, sharing only their small model of parameter size comparable to the small BERT model (117M against 110M parameters). For this reason, we have chosen to compare both small models. We have used as a base the pytorch implemented repository of Huggingface team,

[3]https://github.com/naver/sqlova
[4]https://github.com/lanpa/tensorboardX
[5]https://www.tensorflow.org/guide/summaries_and_tensorboard
[6]https://openai.com/blog/better-language-models/

`pytorch-pretrained-BERT` [7], where the implementation of Google's BERT, but also OpenAI GPT/GPT2, and Google/CMU's transformer-XL model are available as pretrained models.

After analyzing the weaknesses of SQLova paper, we also wanted to try and replace some of the LSTMs in the SQLnet part of the model. For that matter, we have used the original repository of the Transformer [8] and tried to adapt it to our problem.

We will describe in more details the development of the model in the following parts.

3.2 Baseline

As stated earlier, we have selected SQLova's model as baseline model, and it is formed of three parts. Note: the sequence words used have been first pre-tokenized with CoreNLP tokenizer.

3.2.1 BERT

Bidirectional Encoder Representations from Transformers is table-aware contextualized word representations which, unlike other language representation models, was designed to pre-train deep directional representations using both left and right context for all layers [6]. BERT is used for encoding the query together with the headers of the table. The first token [CLS] is a classification embedding. Subsequent tokens [SEP] are used to separate the query ("What is the...") from the headers ("player", "name"). Each token consists of token embeddings, segment embeddings and position embeddings. The output of the encoder is used as input for the seq2SQL model in the NL2SQL layer. For the decoder layer (i.e. right-to-left), additional SQL-specific vocabulary (such as SELECT, WHERE, etc.), "start" and "end" tokens are added between [CLS] and question words separated by [SEP] for the sequence generation. This insures that their positions remain invariant to questions and tables headers. The original pre-trained BERT model trained on a multitude of datasets (i.e. GLUE, SQuAD v1.1, NER, SWAG) is loaded and an additional layer is added and fine-tuned to work for the required task.

3.2.2 Sequence-to-SQL Generation

Although the SQLova team tested three different layers model on top of BERT encoding (shallow layer, decoder layer and NL2SQL layer), our baseline only uses the best performing module NL2SQL-layer (aka SQLova), inspired by SQLNet [4]. At this stage, hidden layers of BERT outputs become inputs of this layer. WikiSQL queries follow the format: "`SELECT agg_op agg_col WHERE (cond_col_1 cond_op_1 cond_value_1) AND (cond_col_2 cond_op_2 cond_value_2) AND...`". The idea is to separate each element of such a query and calculate the following probabilities:

- `SELECT` clause
  - $p_{\texttt{agg\_col}}(i|Q)$ where i is here corresponding to only one of the columns. Here a column attention is used.
  - $p_{\texttt{agg\_op}}(i|Q, col)$ where i is here corresponding to one of the six following classes $\{\texttt{NONE}, \texttt{MAX}, \texttt{MIN}, \texttt{COUNT}, \texttt{SUM}, \texttt{AVG}\}$. A column attention mechanism is used.
- `WHERE` clause
  - $p_{\texttt{cond\_col}}(col|Q)$ where col is one of the given column of the table and Q is the natural language question. Here a column attention mechanism is used, which is a special instance of the generic attention mechanism to compute the attention map on a question conditioned on the column names. Another probability is used to decide the number of `cond_col`.
  - $p_{\texttt{cond\_value}}(value_t|\texttt{cond}_{t-1}, Q)$. Here is used a selection of text span in the question for which is inferred the start and the end positions of such a condition value. $\texttt{cond}_{t-1}$ here represent the t-1 3-tuple $(\texttt{cond\_col}_{t-1}, \texttt{cond\_op}_{t-1}, \texttt{cond\_value}_{t-1})$, following [7] in this particular point.
  - $p_{\texttt{cond\_op}}(i|Q, \texttt{cond\_col})$ where i is here corresponding to one of the three following classes $\{\leq, =, \geq\}$. A column attention mechanism is also used.

[7] https://github.com/huggingface/pytorch-pretrained-BERT

[8] https://github.com/jadore801120/attention-is-all-you-need-pytorch

Once we have these probabilities designed, we can minimize the standard cross-entropy loss (except for $p_{\texttt{cond\_col}}(col|Q)$ where some hyperparameter is used in [4]).

Figure 1 shows the architecture of NL2SQL architecture where LSTM-q and LSTM-h represents encoders for the question and the column headers.

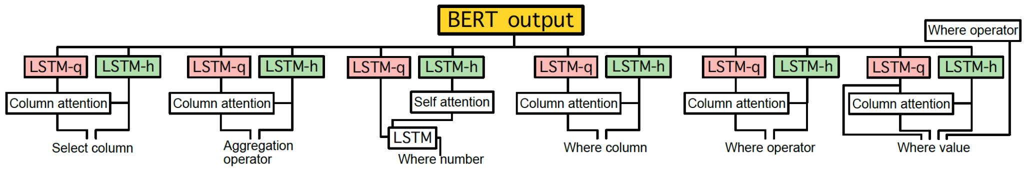


Figure 1: NL2SQL Architecture [11]

3.2.3 Execution-Guided (EG) Neural Program Decoder

After the first two steps, a final EG decoding step is used to remove partial non-executable SQL candidates from the decoding outputs. This includes runtime errors and empty-output errors. EG removes pairs for "SELECT column - aggregate operator" clauses, where the column types are strings and the aggregate operators are numerical (e.g. MAX, MIN, etc.). It also removes pairs of "WHERE column, operator, value" and "SELECT aggregate column", if the query returns an empty answer. In both cases, it then selects the pair with highest joint probability from the remaining pairs.

3.3 OpenAI GPT-2

The baseline model uses BERT pre-model to encode the questions and tables as the input of the model. We have tested a more modern and promising encoding with GPT2, an enhancement of GPT from OpenAI. GPT2 is based on unsupervised learning and transformer architecture code [8] [14]. As GPT2 Large has not been made public yet, our tests will use the released version of GPT2 Small. This had required developing changes to adapt GPT2 to our task.

Compared to its predecessor, the transformer decoder structure of GPT2 has been modified: layer normalization has been moved to the input of each sub-block, and another layer normalization has been added after the final self-attention block. The vocabulary of the BPE encoding has been also enhanced.

However, the dataset change is one of the major changes: GPT used BookCorpus dataset, while GPT2's team created a new web dataset, "WebText", from an oriented quality web scraping of Reddit quality links.

Figure 2 shows the difference between BERT, OpenAI GPT and ELMo:

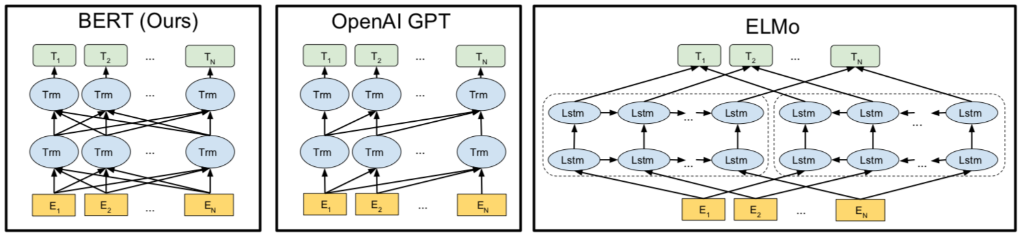


Figure 2: Difference between BERT, OpenAI GPT and ELMo [6]

1. BERT uses a directional Transformer where representations are conditioned on both left and right context in all layers
2. OpenAI GPT uses a left-to-right Transformer

3. ELMo concatenates independently trained left-to-right and right-to-left LSTM to generate features

3.4 Our model

As part of the modifications from the baseline model, we have tried to create a coherent model by testing each part individually. Our aim was to complete testing on at least one change, but ideally all of them. These individual enhancements have been applied to various tasks, however, combining them together for this task has never been done at the time of writing.

3.4.1 Adding and adapting GPT2

3.4.1.1 GPT2 input tokenizer: As describe in [14], the first step to be able to use GPT2 modeling is to adapt the sequence inputs to the GPT2 tokenization system. Byte Pair Encoding (BPE) tokenization has more granularity than the tokenization used in BERT model. It consists of tokenizing sequences to something at a middle ground between characters and word level [1] and a vocabulary length of 50,257, whereas the tokenizer system in BERT is based on the WordPiece embeddings which counts 30,000 words vocabulary [2]. We used the methodology described in [14] : we defined random `<s>` and `<e>` token to delimitate beginning and end of sequences, and we separated headers columns and questions with the token `$`.
The direct consequence was to also increase the max length sequence, as we have naturally more tokens than BERT. We had to slightly decrease the batch size in order to be able to train without compromising the memory of the GPU.

3.4.1.2 GPT2 output change: In the baseline model, the 2 last layers of the BERT transformers output are used to create a context representative embedding for the input of the SQLNet (more exactly, each layer of the transformer has a hidden dimension of size 768; concatenating the two last layers gives us input vectors of size 1536.). In the implementation of the GPT2 we don't have access to all the layers but only the last remaining one which is obtained by passing through a last normalization layer as explained in [14]. Hence, in one of our implementation we tried to change the GPT2 library to have access to the layer before the last. In order to have a comparable layer, we also tried to apply a normalization layer at its top. We then concatenated both layer outputs for each vector as for the output of BERT.

3.4.1.3 GPT2 optimizer: We changed the optimizer for GPT2 fine-tuning to the one used by OpenAI, introducing a system of learning "warmup".

3.4.2 Restructuring SQLNet with transformer

We have implemented a replacement of LSTM encoders in the code for one of the modules of SQLNet, the Select Aggregation probability (module with the poorest accuracy compared to other probabilities), with a transformer structure. Nevertheless, the original implementation of the transformer assumes that the input is a list of tokens and not already a contextual embedding as a BERT nor a GPT2 output model:

$$h_0 = UW_e + W_p \tag{1}$$

Where U is the context vector of tokens, W_e and W_u are the token and position embedding matrices, respectively. We replaced this input layer by:

$$h_0 = W_r h_{pre} + W_p \tag{2}$$

Where W_r is a matrix to allow the change (reduce dimension) of size dimension from a GPT2 or BERT to the transformer hidden dimension, h_{pre} is the output contextualized token sequence of GPT2 or BERT, and h_0 is the input in the first block of the transformer.

4 Experiments

4.1 Data

WikiSQL is a large semantic parsing dataset consisting of 80,654 natural language utterances with corresponding SQL annotations on 24,241 tables extracted from Wikipedia. It was released with the original work by the Salesforce team [5] and has been divided (randomly) into train / dev / test sets for the WikiSQL Competition. However, it is important to note that each table is present in exactly one set.

4.2 Evaluation Method

The primary evaluation metric for ranking model performance for the WikiSQL Competition is the "execution accuracy", e.g. if the result of the model query execution is the same as the test query execution (we can have different query utterances but the same execution result). The "logical form accuracy", measuring the exact similarity with the test query utterance, is also reported as a secondary measure.

The evaluation method provided by WikiSQL does not allow the use of the Execution-Guided (EG) decoder. Another evaluation method, classified as Inference on the leaderboard, developed by Wang (2018) [9] allows the computation.

To improve our model understanding, more granular accuracy measures are calculated for each sub-module of the NL2SQL model, as described on Figure 1. It allows us to monitor if each sub-task is well implemented and modeled. More details regarding the accuracy measures can be found in Table 1.

4.3 Experimental Details

All the LSTMs of SQLova sub-modules have an hidden dimension of 50 for headers and 50 for questions, and 2 layers. The dropout rate is 30%.

For our transformer we used the same hidden dimension, and 6 layers. The inner position wise feed forward inner dimension is 2048. The dropout rate is 10%.

In order to run the baseline model, we use a learning rate of 0.00001 for the BERT fine-tuning, and a learning rate of 0.001 for the NL2SQL, and a batch size of 8 (due to GPU memory limitations). The Execution Guided was activated with a beam size of 4 for search. We used an Adam optimizer. The max sequence length is 222. The baseline ran for 9 epochs with training time close to 8 hours. Our baseline nearly reproduced the train/dev results reported by the SQLova team (see Section 3 for details).

For our model we proceed with a fine-tuning learning rate for GPT2 of 0.0006, and we kept a learning rate of 0.001 for NL2SQL. Due to a larger max sequence length, we decreased the batch size to 7. We also ran it for 9 epochs with training time close to 10 hours.

4.4 Results

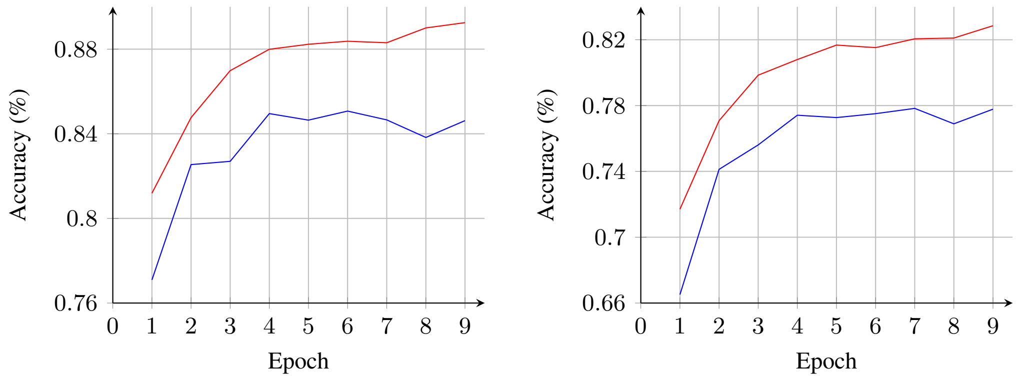


Figure 3: Execution accuracy (acc_x, LHS) and Logical form accuracy (acc_lx, RHS) on the dev set

We have successfully re-implemented the SQLova model and reproduced the performance claimed in their paper: dev results of 82.8% for the logical form accuracy and 89.0% for the execution accuracy (compared with 84.2% and 90.2%, respectively) - using BERT small, compared to their numbers generated with BERT large. Charts showing the details of the evolution of training accuracy can be found in the appendix 5. It has to be noted that the basic evaluation from WikiSQL test script does not make use of EG, producing lower dev and test accuracies. However, projects also report their results with EG, marked as "Inference" and topping the leaderboard.

The best dev logical form accuracy given by our model is 78.0% and the best execution accuracy is 84.9% . Likewise, the best GPT2 model based on the dev set provides 77.5% logical form accuracy and 84.4% execution accuracy. The results for GPT2 show us that despite a performance inferior as BERT, the model still provides an excellent score and proves the robustness of a pre-trained language model + specific task model. In order to find out the reason behind those differences, we tried a few hypotheses:

- We ran the model with only one layer as our 2 concatenated layers output gave us worse results. It seemed that contrary to BERT it is not easy to extract more information from other layers in the transformer architecture, maybe because architecture and order of layers have been changed, as explained above. As a result, we used GPT2 embeddings output of half the size of BERT's which were used by SQLova team, which probably has an impact.
- As the size of GPT2 small model is very similar to BERT, we concluded that the inherent structure of BERT is better due to its bidirectional and particular training protocol with masked language models. The quality of the GPT2 proprietary training data set doesn't seem to compensate the inferiority of the architecture.

Nevertheless, we probably expect to have much better results with large GPT2 and why not a new state-of-the-art, once this model is made available to the public.

Another point to mention is that the transformer testing has not proven itself conclusive and we decided to stop the training early. We expect to have more work to do to make the transformer work properly with this task. For instance, the change of input structure explained above might have to be adapted further.

5 Error analysis

Symbol	Module	BERT (%)	GPT-2 (%)
`acc_sc`	Select Column	97.0	96.6
`acc_sa`	Select Aggregation	90.6	90.7
`acc_wn`	Where Number	97.0	93.4
`acc_wc`	Where Column	95.2	90.5
`acc_wo`	Where Operator	95.5	91.7
`acc_wv`	Where Value	95.3	90.3

Table 1: Best dev set accuracy measures for each module of the model

We have found that a large amount of errors do not come from the model itself, but rather from either the data or the way the accuracy is tested:

- Some of the data was wrongly annotated, causing errors in execution accuracy but also many cases of Selection Aggregation errors (where MIN, MAX or COUNT are wrongly present or for no apparent reasons in the annotated query); a simple (yet, time consuming) solution would be to tidy up the dataset, e.g.:
  Question: What is the power kw@RPM of the 13.180 model?
  Result Predicted (Gold): '127 @ 2400' (1)
  Query Predicted:
  `SELECT Power kW@rpm FROM table WHERE Model = 13.180`
  Query Gold:
  `SELECT COUNT Power kW@rpm FROM table WHERE Model = 13.180`
- The entity name extracted by the model does not match exactly the data in the database (either too much or too little info); a solution would be to introduce the SQL operator `Like % entity %` as a replacement for = in some cases, e.g.:

Question: How many prizes were there at the transamerica?
Result Predicted (Gold): 0 (1)
Query Predicted vs Gold, respectively:
```
SELECT COUNT 1st Prize($) FROM t WHERE Tournament = transamerica
SELECT COUNT 1st Prize($) FROM t WHERE Tournament = The Transamerica
```

- The ordering of `Where` clause generated from the model is different from the annotated one, causing a logical form inaccuracy while the query is perfect; solution would be to add multiple possible queries for examples with more than one possibility, e.g.:
  Question: How many times was the total more than 1, the nation was east germany and silver was more than 1?
  Result Predicted (Gold): 0 (0)
  Query Predicted vs Gold, respectively:
  ```
  SELECT COUNT c4 FROM t WHERE c1 = east germany AND c3 > 1 AND c5 > 1
  SELECT COUNT c4 FROM t WHERE c5 > 1 AND c1 = east germany AND c3 > 1
  ```
- In the case of `COUNT`, the selected column does not matter as long as the rest of the query is correct, causing a somehow unjustified logical form inaccuracy; a solution would be to ignore the column for `COUNT`, e.g.:
  Question: How many tournaments in Texas had a purse higher than 330105.162427?
  Result Predicted (Gold): 1 (1)
  Query Predicted vs Gold, respectively:
  ```
  SELECT COUNT col1 FROM t WHERE col2 = texas AND col3 > 330105.162427
  SELECT COUNT col6 FROM t WHERE col2 = Texas AND col3 > 330105.162427
  ```

6 Conclusion

We have successfully created a model and a tool to run a query on any provided SQL database with a near state-of-the-art accuracy. We have replicated the results described by the SQLova team (i.e. 83.6% for logical form accuracy and 89.6% for the execution accuracy) and showed that replacing the BERT model with the newly released OpenAI GPT-2 did not improve the accuracy, providing only 77.5% logical form accuracy and 84.4% execution accuracy.

The research conducted in this paper highlights the fact that GPT2 (small) is not better than BERT (small) as a pre-model to generate SQL queries from natural language utterances. Both models having very similar number of parameters, we believe that the strength of GPT-2 resides in the large version of the model which has over 1.5bn parameters (i.e. far more than BERT large). Similarly, using a transformer model for the weaker sub-module `Select Aggregation` did not help overcome the errors found with the LSTM.

An interesting application of this model would be to use it for business intelligence. Any user could request a given database without knowledge of the underlying schema or query language and retrieve the required data effortlessly. Plots could then be generated automatically depending on the format of the data received.

Future works for this task could include the ability to train the model on joined tables. However, this would require to crowdsource a more complete dataset as the WikiSQL dataset does not provide the concept of "join". An improvement that we plan to do, however, is the possibility to automatically predict the table name required to run the query. WikiSQL does not require the prediction of the table name (i.e. table-id) since every question is asked for a particular table. However, for a real-world application, it would be more useful to let the model find the table it has to generate the query from.

Acknowledgments

We would like to thank particularly Professor Christopher Manning, the Head TA Abigail See, our mentor for the project Pratyaksh Sharma, and all the other TAs of CS224N for their support and help throughout the quarter.

References

[1] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *CoRR*, vol. abs/1508.07909, 2015. arXiv: 1508.07909. [Online]. Available: http://arxiv.org/abs/1508.07909.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *CoRR*, vol. abs/1609.08144, 2016. arXiv: 1609.08144. [Online]. Available: http://arxiv.org/abs/1609.08144.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

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

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

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018.

[7] L. Dong and M. Lapata, “Coarse-to-fine decoding for neural semantic parsing,” in *ACL*, 2018.

[8] A. Radford, “Improving language understanding by generative pre-training,” 2018.

[9] C. Wang, P.-S. Huang, A. Polozov, M. Brockschmidt, and R. Singh, “Execution-guided neural program decoding,” *CoRR*, vol. abs/1807.03100, 2018.

[10] P. He, Y. Mao, K. Chakrabarti, and W. Chen, *X-sql: Reinforce context into schema representation*, Mar. 2019.

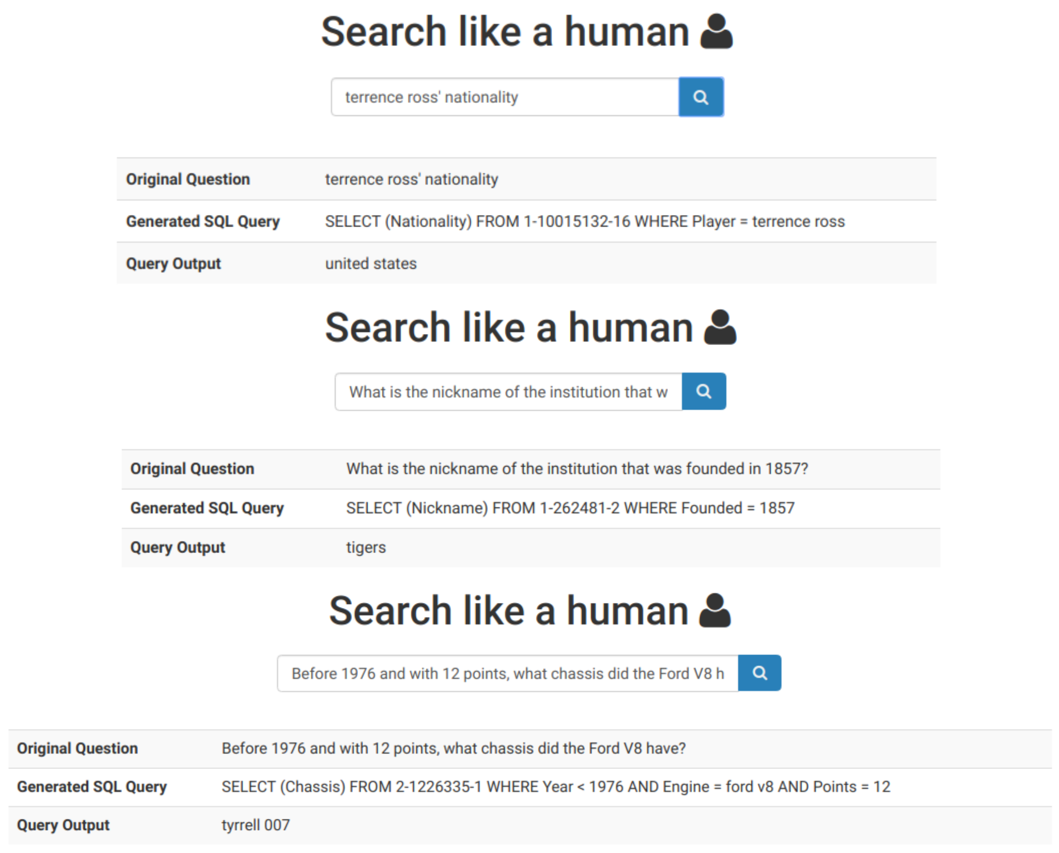
[11] W. Hwang, J. Yim, S. Park, and M. Seo, “A comprehensive exploration on wikisql with table-aware word contextualization,” 2019.

[12] W. Hwang, J. Yim, S. Park, and M. Seo, “A comprehensive exploration on wikisql with table-aware word contextualization,” *CoRR*, vol. abs/1902.01069, 2019. arXiv: 1902.01069. [Online]. Available: http://arxiv.org/abs/1902.01069.

[13] X. Liu, P. He, W. Chen, and J. Gao, “Multi-task deep neural networks for natural language understanding,” *CoRR*, vol. abs/1901.11504, 2019. arXiv: 1901.11504. [Online]. Available: http://arxiv.org/abs/1901.11504.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019.

Appendix



Search like a human

terrence ross' nationality

Original Question	terrence ross' nationality
Generated SQL Query	SELECT (Nationality) FROM 1-10015132-16 WHERE Player = terrence ross
Query Output	united states

Search like a human

What is the nickname of the institution that w

Original Question	What is the nickname of the institution that was founded in 1857?
Generated SQL Query	SELECT (Nickname) FROM 1-262481-2 WHERE Founded = 1857
Query Output	tigers

Search like a human

Before 1976 and with 12 points, what chassis did the Ford V8 h

Original Question	Before 1976 and with 12 points, what chassis did the Ford V8 have?
Generated SQL Query	SELECT (Chassis) FROM 2-1226335-1 WHERE Year < 1976 AND Engine = ford v8 AND Points = 12
Query Output	tyrrell 007

Figure 4: Examples from the live (web) demo on three different test tables

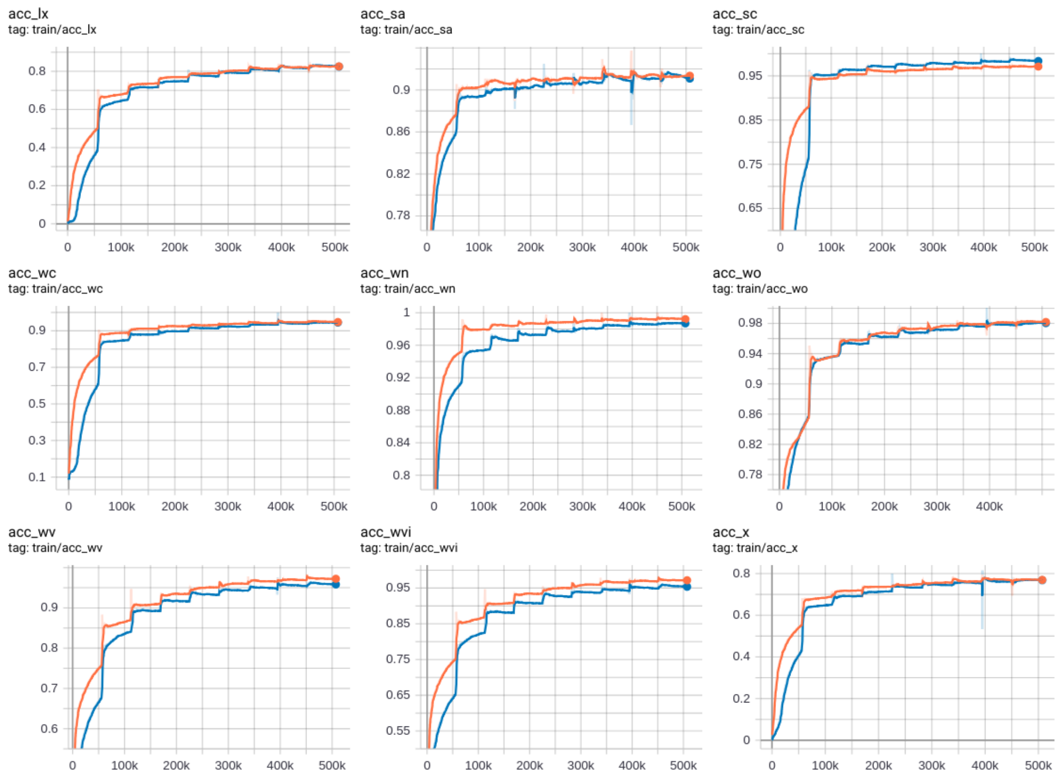


Figure 5: Training accuracy curves for: Select Aggregation (sa), Select Column (sc), Where Column (wc), Where Number (wn), Where Operator (wo), Where Value (wv), Where Value Index (wvi), Execution Accuracy (x) and Logical Form Accuracy (lx). Baseline is represented in orange and GPT2-model in bleu.