Natural Language to SQL Query generation

Isak Axelsson - isaax996,
Eric Dahlgren - erida597,
Jonatan Jonsson - jonjo049,
Jayesh Dinesh Kenaudekar - jayke602,
Nikil Johny Kunnappallil - nikku485,
Anton Lifwergren - antli914,
Gustav Lindahl - gusli687

January 15, 2022

1 Introduction

In this world of fast technological advancement and digitalization, multiple situations require the human to interact with computers. Many applications and services use databases to store configurations, user information and other necessary data to run successfully. Interfacing these databases can be challenging task that requires many resources and domain knowledge. In order to develop software that can extract, alter and insert data into a database, the developer needs prior information about Database Management Systems (DBMS) and their corresponding query languages. One such query language is the Structured Query Language (SQL) [1], which has been the de facto standard language to interface Relational Database Management Systems (RDBMS) for over 40 years. The RDBMS technology has existed for many decades which has led to many fields and industries adopting it, thus these are used in important services such as banking, education and medicine.

For a non-technical user to be able to extract data and interface the RDBMS, Natural Language Processing (NLP) techniques can be used for translation of natural language to SQL queries. This problem is referred to as Text-to-SQL translation and is explored in this paper. Modern NLP architectures that can be used to solve various natural language tasks use deep learning (DL) and neural networks (NN) to train large general models with millions of parameters. This step is called pre-training which is most often followed by another training step where the model architecture adapts to a more specific task, usually referred to as transfer learning [9].

For this report, some general research was conducted to identify existing solutions for Text-to-SQL. This included looking at the WikiSQL leaderboards [14], as well as using search engines such as Google Scholar to explore previous reports and ideas on the topic. From the identified solutions, three different methods were selected to be the focus of this paper. The project group was split into three subgroups, where each group was responsible for implementing one of the selected methods and training the model on the WikiSQL dataset. This approach was deemed the best, since the different models could be evaluated and the most suitable approaches would be further implemented with the focus on using transfer learning to predict SPARQL queries. As such it seemed like a good methodology and the expectation was that it is an adequate way to solve the problem.

2 Theory

This section provides the theory and background of the problem at hand.

2.1 Query Languages

Query languages are means of communicating with various types of databases, similarly to how for example written English can be used by us humans, but with narrower and more scripted use cases. This communication is for most query languages broken down into the following main operations that interface with the underlying database: ”data retrieval”, ”data storing” and ”data modification”. Depending on the database architecture, can these operations take on different syntactic structure, but can for the trained eye often be distinguished and easily compared. This project concerns itsself with the query languages ”SQL” and ”SPARQL”, for which there are example queries for each languages in Figures 1 and 2.

```
SELECT Person.fname, Address.city
FROM Person, Address
WHERE Person.addr=Address.ID
AND Address.state="MA"
```

Figure 1: Example of a SQL query.

```
SELECT ?name ?city
WHERE {
?who <Person#fname> ?name ;
<Person#addr> ?adr .
?adr <Address#city> ?city ;
<Address#state> \MA"
}
```

Figure 2: Example of a SPARQL query retrieving the same information as in Figure 1.

2.2 Transfer learning

In classical supervised machine learning the standard is for a single model trained in isolation on the specific dataset from scratch. This approach requires a large number of training examples and the model performs best for well defined and narrow tasks. On the other hand, *Transfer Learning* is a method that extends this approach by leveraging data from additional domains to train a model with better generalization properties [11]. Over the past 2-3 years there has been rapid advancements in the field of NLP(Natural Language processing) by applying transfer learning models and architectures. This has shown state-of-the-art results on various NLP related downstream tasks like machine translation, question answering, summarization etc. Some well known pre-trained models on large text corpus are BERT, GPT3, ELMo and T5. These advancements along with the ease of integrating these transfer learning methods help us readily use these powerful models in our specific applications and achieve better results over traditional supervised learning methods.

2.3 Baseline

A baseline is an important stepping stone for any technical project. The baseline can be seen as a first solution as it is often the first version to solve a problem. This solution can be used for many things such as communication to make sure everyone is on the same page. Another usage may be to figure out how to allocate resources in the project or as in this paper where it is used as a simple model to compare everything to [4].

As a baseline for this paper, a model based on a paper by Xu et.al. was used [16]. In this model the output from two Long Short Term Memory (LSTM) models is used to predict the different parts of a SQL query. From the dataset the FROM clause is always given and thus does not need to be predicted. This leaves six unknowns for the query: SELECT column, SELECT aggregate, number of WHERE clauses, WHERE columns, WHERE operators and WHERE value. The description on how each of these are predicted can be read below.

Number of WHERE clauses

To get the probability bands for the number of WHERE clauses the following equation is used

$$P_{\#col}(K|Q) = softmax(U_1^{\#col} tanh(U_2^{\#col} E_Q))$$

where E_Q is the output from the LSTM (for the embedded questions) in the last layer for each timeslot, $U_1^{\#col}$ and $U_2^{\#col}$ are trainable matrices of size $d \times (N+1)$ respectively $d \times d$, where d is the hidden dimension of the LSTM and N is the maximum amount of where clauses (in this case set to 4). The model then chooses the top-K columns.

WHERE columns

To get the probability bands for the WHERE columns the following equations are used

$$P_{wherecol}(col|Q) = \sigma(u_c E_{col} + u_q E_{Q|col})$$

$$E_{Q|col} = E_Q w$$

$$w = softmax(E_{col}(WE_Q)^T)$$

where E_{col} is the output from the LSTM (for the embedded columns) in the last layer for each timeslot, u_c and u_q are trainable vectors of dimension $d \times 1$ and W are a trainable matrix of size $d \times d$.

WHERE operators

To get the probability bands for the WHERE operator the following equation is used

$$P_{op}(i|Q, col) = softmax(U_1^{op} tanh(U_c^{op} E_{col} + U_q^{op} E_{Q|col}))$$

where U_1^{op} is a trainable matrix of size $d \times o$, where o is the number of operators (here $o = 3$), and U_c^{op} and U_c^{op} are trainable matrices of size $d \times d$.

WHERE value

For the value part a simple randomizer is used where it picks at random $argmax(P_{\#col})$ words from the question.

SELECT columns

To get the probability bands for the number of SELECT columns the following equation is used

$$P_{selcol}(i|Q) = softmax(u_a^{sel} tanh(U_c^{sel} E_{col} + U_q^{sel} E_{Q|col}))$$

where u_a^{sel} is a trainable vector of dimension $d \times 1$ and, U_c^{sel} and U_q^{sel} are trainable matrices of size $d \times d$.

SELECT aggregate

To get the probability bands for the number of SELECT aggregators the following equation is used

$$P_{agg}(i|Q, col) = softmax(U^{agg} tanh(U_a E_{Q|col}))$$

where U^{agg} is a trainable matrix of size $d \times a$, where a is the amount of aggregators (here $a = 6$), and U_a is a trainable matrix of size $d \times d$.

Loss function

When the model is training it tries to minimize some function. For the baseline this function is six different functions that are added together. For four of the probability bands it will be the normal cross-entropy and for the last it will be the following equation

$$loss(col, Q, y) = -(\sum_{j=1}^{C}(\alpha y_j log(P_{wherecol}(col_j|Q)) + (1-y_j) log(1-P_{wherecol}(col_j|Q))))$$

here α is a hyper-parameter that was set to 3 and C is the number of columns. Note that the WHERE value is not part of the loss function.

2.4 Transformers

Transformers refers to both the general neural network architecture aimed at solving various NLP tasks [13], as well as the library of carefully chosen and publicly available transformer based architecture implementations [15]. This section will cover the former.

Abstaining from using previously successful methods such as Long Short-Term Memory and maybe the most well known technique Recurrent NN's, the transformer architecture is based on attention[13], more precisely self-attention. This mechanism aims to represent a given sentence by relating all words (or tokenized versions of the words) and their positions in the sentence. This approach has been used in other architectures, and other areas of machine learning for that matter, before but not as the sole mechanism in solving NLP problems. The usage of this can be seen in the structural overview of the transformer in Figure 3.

Making use of an encoder (left in Figure 3) and a decoder (right in Figure 3), two commonly used divisions of models in sequence transduction, the transformer, very much simplified, maps input symbol sequences of some kind to an output sequence of symbols of some other kind. This could be for example a sequence of English words making up an actually meaningful sentence, that the transformer maps to a sequence of SQL keywords and parameters that make up a meaningful SQL query.

The encoder can abstractly be seen as a "general language meaning extractor", that outputs (the top of the encoder in Figure 3) the continuous representation

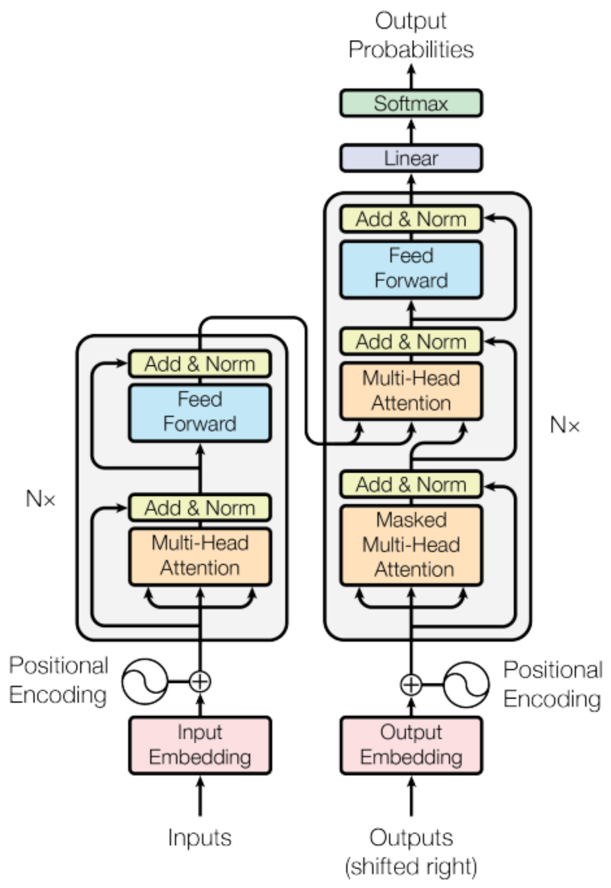

Figure 3: The Transformer - model architecture. Figure taken from [13]

of an input which it, the encoder, believes represents the context and meaning of the input most accurately. This output in its "raw" form is not feasible to interpret as a human, but rather a representation tailor-made for the specific transformer's decoder. The decoder can in some cases [3], for specific implementations of the transformer, be reused or interchanged to solve specific NLP tasks given the output from the encoder.

Following this section are variants of the transformer architecture that are covered in this project.

2.4.1 T5- Transformer

Text-to-text Transfer Transformer model, or T5, is a pre-trained encoder-decoder transformer model trained on the colossal cleaned crawled corpus(C4) [10]. The pre-training was done on multi-task mixture of unsupervised and supervised task for which each task is converted into text-to-text format.

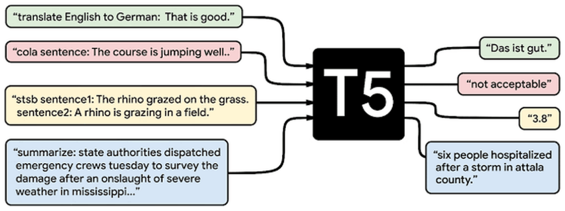
"translate English to German: That is good."
"cola sentence: The course is jumping well.."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi..."
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 4: The T5 Transformer model. Taken from [10]

The various task that T5 can perform is machine translation, question-answering, sentiment analysis, text summarization etc.

T5 uses relative scalar embedding. T5 model has been trained on different sizes, the one which is we use in this project is the T5-base, which has 220 million trainable parameters with 12 Attention heads.

2.4.2 SQLOVA

SQLOVA is a state-of-the-art model for translating natural language questions into SQL. The model uses the output of a table-aware BERT encoder as word embeddings for its `NL2SQL_LAYER`. Wonseok Hwang et al. describes how they achieved an accuracy of 90% on the WikiSQL dataset[6]. This report used the same structure but a slimmer table-aware BERT encoder due to limited resources.

The table-aware BERT encoder extend BERT by encoding the question query with the headers of the table in the following way[6, 2]:

```
[CLS], question tokens, [SEP], header 1 tokens, [SEP], header 2 tokens,
[SEP], ...  , [SEP], last header tokens, [SEP]
```

where [CLS] and [SEP] are special tokens for classification and context separation. The given English question is split into tokens forming the *question tokens*. The *headers* are the headers from the SQL table. Note that [SEP] is used to separate the question and the headers. An overview of the table-aware BERT-encoder can be seen in Figure 5

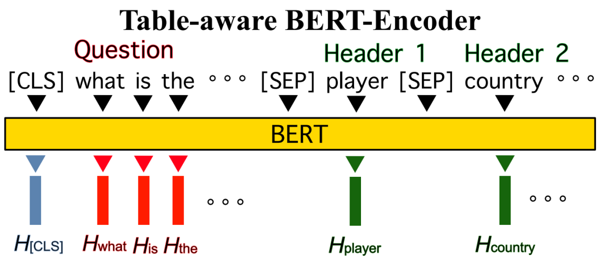
Table-aware BERT-Encoder
Question
Header 1
Header 2
[CLS] what is the ∘∘∘ [SEP] player [SEP] country ∘∘∘
BERT
$H_{[CLS]}$ H_{what} H_{is} H_{the}
H_{player}
$H_{country}$

Figure 5: An illustration of the table-aware BERT encoder scheme taken from [6]. The output is color-coded: blue for the [CLS] token, red for question tokens and green for the header tokens.

The NL2SQL-layer uses the table-aware BERT encoder's output as word-embedding vectors[6]. The word-embedding vectors are feed into six different modules. In each module the word-embedding vectors are encoded with LSTM-q (question encoder) and LSTM-h (header encoder) and column attention is employed. The modules do not share any parameters and the six resulting output values are used to generate a SQL-query [6]. This can be seen in Figure 6.

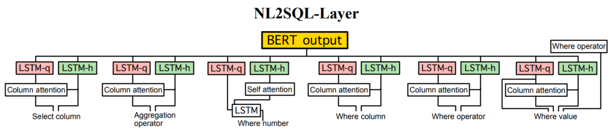

Figure 6: An illustration of the NL2SQL-Layer scheme taken from [6].

3 Method

Three models were selected to be investigated during the project, one baseline and two advanced models. SQLuwu was implemented as the baseline and T-5 and SQLOVA as the two advanced models. This section describes how the models have been adopted and any necessary pre- or post-processing implementation.

3.1 Baseline

As a baseline for this paper the model described in section 2.3 was used. This was implemented in Python with PyTorch. For the model to work it has to get a question and table-id. With this table-id it will when get columns given the table. The model then takes as an input the question and the corresponding columns which it then tokenizes and passes it through a pre-trained GloVe embedding.

During training, the model was trained for ten epochs in batches of 64 questions. After each epoch the model was evaluated on the evaluation data. When all epochs have passed, the best accuracy of the model on the evaluation data after each epoch was compared and the best model chosen. This model then performed the final evaluation on the test data.

3.2 T5 Transformer

In our T5 implementation, we address the task of Natural language to SQL Query generation as a Sequence2sequence problem [8]. Since our T5 model is an encoder-decoder architecture, we feed our inputs ie: Natural language query to the encoder and the corresponding output SQL Query to our decoder.

We approach the problem as specified in the paper[8]. We take the inputs for T5 model as question and schema from the WikiSQL Dataset separated by a SEP token.Note: each column name in schema is also separated by a SEP token. Finally the question and schema is concatenated and fed into the T5 model for training along with the correct SQL query as label.

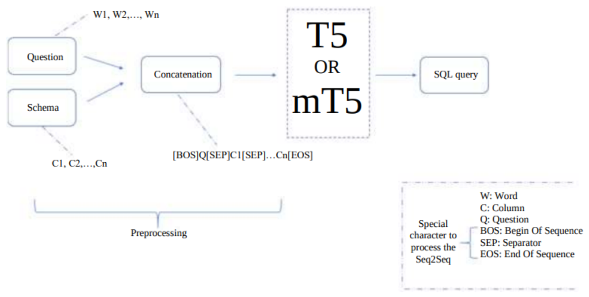

Figure 7: Architecture for using T5 over WikiSQL [8]

Every input string that goes into the T5 model is pre-processed and tokenized using the T5 tokenizer and padded to a predefined max sequence length. The tokenizer converts the concatenated input string into tokens or words which is then vectorized into its unique integer ids. This transformed data is fed into our T5 model to be fine tuned on our pre-processed WikiSQL training data-set. Finally, we measure the exact match accuracy or If accuracy on the test dataset and the corresponding results can be found in the Table 1 in Section 4.

Similarly for QALD(Question Answering over Linked Data) we applied transfer learning by replacing schema by keywords as inputs to the model to generate sparQL queries. We preprocess the training data for sparQL by replacing the links in the queries by unique integer ID's which we decode separately once the model generates the sparQL query. This entire study over QALD dataset was done to explore the limits of transfer learning on small scale dataset.

3.3 SQLOVA

As our final approach, the SQLOVA architecture [5] was adopted as the "state-of-the-art" solution for our problem of translating natural language into SQL-queries. This model was choosen due to its prior good performance on this exact task, which can be seen on the WikiSQL github [14] on the "Supervised via logical forms" leaderboard. Here the model can be confirmed to give a great resulting accuracy, being only a couple percentage points behind the current leader (which was added to the list as of 2021), though it should be mentioned this leaderboard is based on using a larger Bert parameter set than that of this project.

The pre-processing of the WikiSQL data set is essential because of how the initial layer is configured and how the data is passed and used in the evaluation layer of the model. By looking at an example of an input data point provided by the WikiSQL data set, and what it looks likes after the pre-processing step is applied, it is more obvious what is modified during this process. An example of this process is visualized in Figures 3.3 and 3.3, the former showing the original and the later the result of the pre-processing.

```
 {
"phase":  1,
"table_id":  "1-10015132-16",
"question":  "What is terrence ross' nationality",
"sql":  {"sel":  2, "conds":  [[0, 0, "Terrence Ross"]], "agg":
0}
}
```

Figure 8: Example of a SQL query provided in the WikiSQL data set.

```
 {
"table_id":"1-10015132-16",
"phase":1,
"question":"What is terrence ross'nationality",
"question_tok":["What","is","terrence","ross","'","nationality"],
"sql":{"sel":2,"conds":[[0,0,"Terrence Ross"]],"agg":0},
"query":{"sel":2,"conds":[[0,0,"Terrence Ross"]],"agg":0},
"wvi_corenlp":[[2,3]]
}
```

Figure 9: The pre-processed version of the SQL query in Figure 3.3.

The major differences are firstly that the English sentence is represented both as a continuous string (the "question" entry) as well as in its tokenized form (the "question_tok" entry). The final json entry ("wvi_corenlp") in Figure contains a list of start and stop indices of the important "entity-spans" in the sentence (where the corresponding entities are stated as the third argument in the "conds" entry). In Figure this refers to the boundary indices of the name "Terrence Ross" in the sentence. This entry is precomputed using the standford python package CoreNLP [7].

To separate the functionality of the SQLOVA pipeline, where there separate files created for the three major steps in the architecture. These steps are "training", "generating prediction queries" and "evaluation epoch results". In the original SQLOVA implementation was the file "train.py" responsible for all these steps, which did not seem intuitive nor made it possible to run using our available hardware. These steps can now be run independently using our solution, but requires manual intervening for selecting the best epoch.

In order to avoid over fitting the WikiSQL data set it is split in to three sets: a testing set, development set and training set. The model is trained on the whole training set and evaluated on the development set for each epoch. The results on the development set is compared and the model with best score is picked as the final version of the SQLOVA model.

4 Results

This section presents the results of running the three different models on the WikiSQL [17] benchmark, as well as the results of using the best performing model for the Text-to-SPARQL task.

4.1 WikiSQL

All competing models on the leaderboard stated on [14], which are using the WikiSQL data set, are calculating the "WikiSQL benchmark", which is why it is also recorded by the models used in this project. This benchmark consists of: "Dev logical form accuracy" (predicting the correct query) and "Dev execution accuracy" (extracting the correct information from the data base). These are separated due to the model being able to extract the correct information from the data base even though the query is partly incorrect in quite a few cases, hence this score often being higher. In Table 1 below are the resulting accuracies from all models.

Model	Dev logical form accuracy	Dev execution accuracy	Test logical form accuracy	Test execution accuracy
Baseline	00.23%	00.55%	00.21%	00.59%
T5	60.53%	-	58.96%	-
SQLOVA	71.92%	78.28%	71.61%	78.13%

Table 1: Results on the WikiSQL benchmark for all models.

In Table 1, can the results of the three models be compared on the WikiSQL benchmark task. Both the T5 based model and SQLOVA perform significantly better than the baseline and SQLOVA outperforms the T5 model by more than ten percentage points for logical form on both the test and validation set.

4.1.1 Baseline

Since the baseline is a sequence-to-set predictor, the individual parts of of a SQL query and their prediction accuracies can be presented.

Dataset	Select	Aggregator	# Where	Where columns	Where op	Where val
Dev	61.62%	84.14%	90.41%	7.02%	76.53%	1.89%
Test	32.67%	44.60%	47.93%	3.72%	40.57%	1.00%

Table 2: Results on the WikiSQL benchmark for the baseline when looking at individual parts that make up a query result.

4.1.2 SQLOVA

SQLOVA was evaluated on dev dataset for each epoch in order to avoid over fitting. Below are the results from each epoch:

Epoch	Dev logical form accuracy	Dev execution accuracy
Epoch 1	69.20%	62.20%
Epoch 2	78.28%	71.92%
Epoch 3	64.55%	55.27%
Epoch 4	63.76%	54.80%

Table 3: Results on the WikiSQL benchmark for the SQLOVA when selecting the number of epochs to train on.

The model performed best after two epochs of training when evaluated on the wikiSQL development data set, therefor that model was selected as the final SQLOVA model.

5 Discussion

Having completed our project on Natural language to SQL query generation, we wrap up by first recapping some of our most significant findings. Our results provide some high-level perspective on which part of our methods might be more or less promising. To conclude, we also outline some topics that might provide effective approaches to further improve our models.

5.1 Baseline

As shown in Table 1, the baseline does not perform very well, and one can argue that it does not serve well as a baseline for the WikiSQL task. The model used for the baseline is simple in a sense that it uses relatively outdated NLP techniques instead of transformers. However, a baseline should also perform reasonably well so that conclusions can be drawn regarding the performance and results of the more advanced comparing models.

However, in Table 4.1.1 the baseline looks more promising, and the model even has decent accuracy when predicting certain parts of the query. The reason that the logical form accuracy and execution accuracy were so low is that those accuracies essentially means the entire predicted query needs to be correct. As such, the model can never have a higher overall accuracy than the worst accuracy for the individual parts. Since our baseline model predicted 1.89% and 1.00% correct for the value part of the where clause, it is reasonable that the overall accuracy is not higher.

As mentioned in Section 2.3, the model took a random subset of the natural language question for predicting the value part which is why the accuracy is so low. The model that we based our baseline on used a more complex method for predicting the value, and it was determined during the project that due to the time constraints of the course we would implement a simpler approach to be able to focus our efforts more on other parts of the project.

5.2 Comparing T5 and SQLOVA

Our text-to-text framework provides a simple way to train a single T5 model on a wide variety of text tasks using the same loss function ie. sparse categorical cross entropy and encoding-decoding procedures. We showed how this approach can be successfully applied to sequence to sequence task such as Natural Language Query(NLQ) to SQL generation. We found that the model inherently learns the natural language query semantics and its representation to map into SQL query.

The SQLOVA model displayed a performance of accuracy comparable to other low performing models on the WikiSQL leaderboard [14]. If the pre-trained parameters for BERT-large would have been used instead of a BERT-base then the accuracy would have been expected to be the same, or at least close to, SQLOVA's official score on the leaderboard. This was however not possible due to the fact that BERT-large would have been too computationally heavy, given our available resources.

Comparing the two, we achieved a logical form accuracy of 59% and 72% for T5 and SQLOVA respectively on the test set. The later has higher accuracy compared to T5 base, which can be partly due to SQLOVA using its NL2SQL layer on top of BERT to generate each SQL Query clause separately whereas T5 base model generates the entire SQL query in one step from its decoder stack.

5.3 Dataset size

We compared the test accuracy of the T5 base transformer on two different data set sizes. One with 1000 examples and the other on the entire training set($\sim$10,000 examples). We see how using the later increased the accuracy by 19%, this when both models were trained for total of 2 epochs.

5.4 Transfer learning for SPARQL translation

Finally, we explore the limits of our Transformer pre-trained models by applying transfer learning on QALD (Question Answering over Linked Data) to generate SPARQL queries. Below we discuss two approaches that were considered and researched, that both tries to achieve this goal with different approaches.

The first approach was based on fine tuning our T5 base transformer, trained on the SQL generation task, to predict SPARQL, but having done the fine-tuning we noticed the training set for SPARQL was very limited to generate accurate SPARQL queries although the generated queries did predict parts of query correctly such as the entities and the syntactic structure. Further, we studied how we can improve the task by augmenting the model with subject identification such as entities using NER (Named entity recognition) and relation classification given an English query. The combination could yield better overall accuracy, downsizing the need for larger data-set.

Our second approached aimed to use transfer learning on the most promising model, SQLOVA, by pre-generating a translation of the SPARQL data set QALD-9 to a SQL data set with the same structure as in WikiSQL. Such a translation is proposed by BERT-QA[12], and would enable us to work around the need of retraining the model to predict queries in the new language SPARQL (which would be counter-productive given the language specific implementation nature of SQLOVA), and essentially keeping the model isolated from SPARQL all together. This solution should in theory work, but after attempting to replicate it, it was obvious that we needed more development time than what we had. But maybe more importantly, one could argue that this solution is not representable of what a translation from natural language to a generic query language would entail, since languages can differ massively in structure and syntax, and that this BERT-QA solution is to narrowly aimed at specifically SPARQL to SQL translation. That this second approach is comparing a generic query language's conformity to the SQL structure.

5.5 Improvements by Scaling

If we were to scale our models further, "scaling" meaning for example training the model on more data, initializing the models with more parameters or using an ensemble of models. An ensemble of models can provide substantially better results than a single model, such as the mT5(multilingual T5) or the ExT5 (Extreme Multi-task scaling) models, which have shown to yield better results compared to its base counterparts. Although scaling the models would come at the cost of requiring additional computational power, the improved accuracies might still outweighs these costs in potential future applications using these models. Other significant improvements are for example using Association Rules and using Execution Guided inference methods(EG), since it is not easy for model to generate perfect SQL queries and these two improvements help alleviate this.

By scaling the SQLOVA model by using BERT-large instead of BERT-base and utilizing EG it has been recorded on the WikiSQL leaderboard that the model achieved a 90% execution accuracy on the test set [14].

References

[1] Don Chamberlin. "SQL". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 2753–2760. ISBN: 978-0-387-39940-9. DOI: `10.1007/978-0-387-39940-9_1091`. URL: `https://doi.org/10.1007/978-0-387-39940-9_1091`.

[2] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[3] Anthony Gillioz et al. "Overview of the Transformer-based Models for NLP Tasks". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. 2020, pp. 179–183. DOI: `10.15439/2020F20`.

[4] Jill A-C Hardash. "Why is a technical baseline important on a non-engineering technical project?" In: *2010 IEEE Aerospace Conference*. 2010, pp. 1–8. DOI: `10.1109/AERO.2010.5446876`.

[5] Wonseok Hwang et al. *A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization*. 2019. arXiv: `1902.01069 [cs.CL]`.

[6] Wonseok Hwang et al. "A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization." In: (2019). URL: `https://login.e.bibl.liu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsarx&AN=edsarx.1902.01069&lang=sv&site=eds-live&scope=site`.

[7] Christopher D Manning et al. *CoreNLP*. `https://github.com/stanfordnlp/CoreNLP`. latest commit: 2021.

[8] Youssef Mellah et al. "SQL Generation from Natural Language: A Sequence-to-Sequence Model Powered by the Transformers Architecture and Association Rules". In: *Journal of Computer Science* 17 (May 2021), pp. 480–489. DOI: `10.3844/jcssp.2021.480.489`.

[9] Shuteng Niu et al. "A Decade Survey of Transfer Learning (2010–2020)". In: *IEEE Transactions on Artificial Intelligence* 1.2 (2020), pp. 151–166. DOI: `10.1109/TAI.2021.3054609`.

[10] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: `1910.10683 [cs.LG]`.

[11] Sebastian Ruder et al. "Transfer Learning in Natural Language Processing". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019. DOI: `10.18653/v1/N19-5004`. URL: `https://aclanthology.org/N19-5004`.

[12] Muhammad Saleem and SukruthRam. *BERT-QA*. `https://github.com/dice-group/BERT-QA`. 2021.

[13] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

[14] Richard Socher Victor Zhong Caiming Xiong. *WikiSQL*. https://github.com/salesforce/WikiSQL. latest commit: 2021.

[15] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6.

[16] Xiaojun Xu, Chang Liu, and Dawn Song. “SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning”. In: *CoRR* abs/1711.04436 (2017). arXiv: 1711.04436. URL: http://arxiv.org/abs/1711.04436.

[17] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. In: *CoRR* abs/1709.00103 (2017).

A Contributions

In addition to the work which we produced together as a group such as planning the project, writing the report and creating the two presentations, have we below summarized the main individual contributions to the project.

Jonatan Jonsson:
Me and Anton Lifwergren worked as a pair for the majority of the course, mainly on implementing and assessing state-of-the-art NLP models for NL to SQL translation. I firstly found the X-SQL model to be a good fit, so I spent quite a long time researching the transformer architecture and implementing this strategy. This was initially attempted by creating it more or less from scratch (using the standard BERT model from Transformers) and applying the modifications found in the X-SQL paper. This required too much work so then I tried to use the pre-implemented version, but the repo was outdated and unusable given the instructions on how the repo was structured. I then resorted to a new model, SQLova, which is similar in performance and architecture. This worked better, and we implemented this for Google Colab (with a few modifications from the given repo), and trained the model and got good accuracy. I then investigated the possibility of transfer learning for this model, which theoretically would be possible (e.g. through means similar to the ones proposed in bert-qa) but this also proved difficult to follow, but gave me insights into how this could be used for this purpose. I was part of, and spoke on, the two presentation in the course.

Anton Lifwergren:
It got a bit long when I wrote so here is a summary: Worked in pair with Jonatan Jonsson, started but did not finish implementing X-SQL, implemented SQLOVA, started with transfer learning but did not reach any results, talked and contributed to the two presentations (mid and end).

Here is a more in depth take:
Me and Jonatan Jonsson worked together on all assignments with a few sessions where we did not directly worked together. For example reading material and some hours of training the model. So in the following when I write ”I” I also mean that Jonatan was apart of it as well. I started working on X-SQL, which looked at a promising model to implement. But after trying to implement it from ”scratch” several problems occurred as discussed on the meetings. This took a lot of time and I realized that it would take more time than what I had. So the second approach was to use X-SQL existing implementation, but their git repo was outdated and hard to use. While working on this I started to look for other models on the wikiSQL leaderboard. SQLOVA was found and their implementation was possible for us to use with some changes. The major tasks were some paths was needed to change, some files were missing and it could not run on our computers GPU. With some digging the paths and the files were found. The GPU problem was solved with google colab and some modifications

on the implementation. The model was trained and tested. SQLOVA also provided an evaluation function which was used for evaluating the Baseline and SQLOVA. I then started to investigate how to perform transfer learning, but we did a clear miss calculation on how difficult it would be. First we had problem finding a data set, then we were unsure what approach we should take since the gold labels change so much between wikiSQL and QALD-9. The tips you Cyrille gave on BERT-QA seemed promising but there were to many problems running the files to be able to have confident in solving it with the time left. It however gave an insight in how it could have been implemented. I also spoke and contributed to the two presentations.

Nikil Johny Kunnappallil:
During the project me and Jayesh worked together on the implementation of transfer learning on T5 transformer model. Since we had previously worked on transformers we started with reading papers and understanding the T5 model. We both did research on papers which were having related works and came up with the current model. Since we were sitting together all the time during work, hence my contributions were mainly on pre-processing dataset, training and evaluating the accuracy of the generated output. The coding part was done in pytorch. Even though we were not able to get good results for Sparql, we were still able to generate some results which looked similar to Sparql query by replacing table column's of schema with keywords of QALD dataset. Training the model on different sizes of data and evaluating the accuracy was done together and it gave a lot of insights for me personally. I was also part of writing the report and the final presentation.

Eric Dahlgren:
Contributed to the project by writing code for the baseline system where the whole system was written from scratch with inspiration from the SQLNet [16] paper. Me, Gustav and Isak created the baseline system together in a mob programming manner, where all of us attended every session with only a couple of exceptions. Every session we would switch whoever was writing code and screen sharing for equal learning opportunities. Even though the results were not great and the results did not really provide a good comparable baseline, it was a valuable learning experience.

Gustav Lindahl:
Contributed in writing the code and chosing model for the baseline with Eric Dahlgren and Isak Axelsson. This code was written from the ground with the help of PyTorch and Numpy. I also wrote and presented the baseline part for the report and presentation. The result for the baseline was not best, mainly because of the randomization of the value part in the where clause. Despite this I have learnt a lot of both working in bigger AI projects and many new technologies in NLP that I will carry on in future work.

Isak Axelsson:
I mainly worked with Eric Dahlgren and Gustav Lindahl throughout the entire

process of creating our baseline model. The model was based on SQLNet, so during implementation we took turns writing code, while we all went through the report written about SQLnet and the corresponding code for it and took inspiration or made adjustments where needed to fit our purpose. I also wrote parts of the report, including contributions to the introduction and the results. Additionally, I prepared and presented the introduction during the final presentations.

Jayesh D. Kenaudekar:
I along with my partner Nikil J.K, worked on the implementation of T5 based transformer model. Having both worked on transformer architecture from scratch in our previously partnered course, in this project we aim to use large transformer based pretrained model - T5 base which was trained on a large English corpus and we exploit the limits of transfer learning for our intended downstream task, i.e Natural language to SQL query generation. The motivation to use T5 was inspired by its encoder decoder architecture that are basically the MHA blocks of the transformer model. The entire architecture was implemented step by step gathering insights from various resources on the web and the code was written in pytorch. During all times both of us dedicated an equal amount of time towards gathering the dataset, preprocessing it as required by the T5 model and finetuning the model. Mainly my contribution was towards bringing in ideas and putting it into code. All strategies were thoroughly discussed with my co partner and implemented in practice such as evaluation metric and transfer learning on QALD dataset for Sparql query generation.