Exploring the Potential of Using Transfer Learning with Google T5 Model for Text-to-SQL Tasks

Anonymous UBB CVDL submission

Paper ID *****

Abstract

The recent advancements in natural language processing (NLP) have led to the development of powerful pre-trained models such as Google T5. These models have been shown to achieve state-of-the-art performance on a wide range of NLP tasks, including text classification, machine translation, and language understanding. In this paper, we explore the potential of using transfer learning with the Google T5 model for text-to-SQL tasks. Text-to-SQL is a challenging task that involves generating a SQL query from a natural language text. It requires a deep understanding of both natural language and SQL, making it a perfect candidate for transfer learning. We fine-tune the pre-trained T5 model on a text-to-SQL dataset and evaluate its performance on several benchmarks. Our paper shows that transfer learning with Google T5 is a promising approach for text-to-SQL tasks, that could be improved further to reach the current state-of-the-art results.

1. Introduction

1.1. Context

The text-to-SQL task is a natural language understanding task that involves generating a structured SQL query from a natural language text. This is often known in literature as natural language interface for database (NLIDB). This task requires a deep understanding of both natural language and SQL, making it a challenging task for traditional NLP models. However, recent advancements in deep learning and pre-trained models have led to significant improvements in text-to-SQL performance.

One of promising approaches for text-to-SQL is using transfer learning with pre-trained models, such as Google T5 [4]. Transfer learning is a technique where a pre-trained model is fine-tuned on a specific task, allowing it to adapt to the new task while retaining the general language understanding capabilities learned during pre-training. This approach has been shown to be effective for a wide range of NLP tasks, including text classification and machine translation.

Google T5 is a neural network model that was trained on a diverse range of tasks using the text-to-text transfer learning framework. This allows T5 to be fine-tuned for a wide variety of tasks, making it a suitable candidate for text-to-SQL.

1.2. Motivation

If this problem is solved, it's going to be widely useful because the vast majority of data in our lives is stored in relational databases. In fact, healthcare, financial services, and sales industries exclusively use the relational database. Also, writing SQL queries can be prohibitive to non-technical users. Therefore, enabling people to directly interact with large-scale enterprise databases using natural language or voice has the potential to be a real game-changer for the non-technical persons.

1.3. Objectives

The final goal is to create a robust system capable of converting database queries expressed in plain English to SQL queries with an acceptable accuracy. Concretely, in order to explore the possibility of achieving this using our proposed approach, we want to fine-tune the pre-trained Google T5 model on a text-to-SQL dataset, such as WikiSQL, and evaluate its performance on several benchmarks. Then, we want to compare the performance of the fine-tuned T5 model with the previous state-of-the-art models on text-to-SQL tasks and demonstrate the potential of transfer learning with T5.

1.4. Original contribution

The main original contribution of this paper is to demonstrate the potential of using transfer learning with the Google T5 model for text-to-SQL tasks. Also, we explore the potential of improving our current model by changing certain aspects of it, such as the optimizer we are using, certain hyper-parameters (e.g., the batch size, the number of epochs), etc.

2. Literature review

The scope of this section is to offer the reader an overview over the literature in the topic of text-to-SQL tasks. We will go over the advantages and disadvantages of the most common used datasets, and over several approaches used for this problem.

2.1. Datasets

The first text-to-SQL datasets, WikiSQL and Spider, were introduced in 2017 by Zhong et al. [7], respectively Yu et al. [6], in 2018. Since then, several other datasets have been introduced, including the ATIS and NL2SQL datasets. However, most of the current state-of-the-art models still prefer to use the WikiSQL and Spider datasets, which is why we will focus mainly on these two. Moreover, in our approach for this paper we will use the WikiSQL dataset. These datasets have been widely used to evaluate and improve text-to-SQL models.

2.1.1 WikiSQL

The dataset consists approximatively 80,000 questions and corresponding SQL queries for Wikipedia tables, making it one of the first and largest datasets for this task. Each question is paired with its corresponding SQL query and an accompanying table. The SQL queries in the dataset are relatively simple and cover a wide range of SQL keywords and clauses. The dataset is divided into a training set, a development set, and a test set. The dataset has been widely used to evaluate and improve text-to-SQL models. One important aspect to mention here is that WikiSQL constrained the problem by two factors: each question is only addressed by a single table, and the table is known. This constrained setting has guided research to focus on the core elementary problem. Even though the scope is constrained, the dataset is still very challenging because the tables and questions are very diverse. Notably, there are about 24K different tables associated with this dataset.

This is the dataset that we will also use for the approach proposed in this paper.

2.1.2 Spider

The Spider dataset is a large-scale human-labeled dataset used for the text-to-SQL task. The dataset contains over 10k complex questions and corresponding SQL queries for 200 different databases, making it one of the largest datasets for this task. Each question is paired with its corresponding SQL query and an accompanying table. The SQL queries in the dataset are relatively complex and cover a wide range of SQL keywords and clauses, including subqueries and joins. The dataset is divided into a training set, a development set, and a test set. The dataset has been widely used to evaluate and improve text-to-SQL models, especially for cross-domain text-to-SQL task.

2.2. SQLNet model

This model is representative for the seq2seq approaches for the text-to-SQL task. One of the most representative articles that proposed the Seq2Seq approach for text-to-SQL task is the one written by Xiaojun Xu. in 2017 [5], which is why we also chose to present this model in our literature review section.

Sequence-to-sequence (Seq2Seq) is a neural network architecture that is commonly used for natural language processing tasks, such as text-to-SQL. The Seq2Seq model consists of two main components: an encoder and a decoder. The encoder takes in a sequence of input words, such as a natural language question, and generates a fixed-length vector representation of the input. The decoder takes in this vector representation and generates a sequence of output words, such as a SQL query.

SQLNet uses a Seq2Seq model with a novel attention mechanism that is specifically designed for the task of text-to-SQL generation, which is one of its main advantages compared to previous arhitectures. The attention mechanism allows the model to focus on different parts of the input text and the SQL query template at different stages of the generation process, which helps to improve the accuracy and fluency of the generated SQL queries. However, the model had also a few advantages and improvement points tackled in the future arhitectures, such as not being able to handle more complex and longer queries (e.g., join queries, since it was trained on WikiSQL), it might not be generalize well to some other database systems, it might not be able to handle language ambiguities and variations in natural language text.

The model achieved state-of-the-art performance on the benchmark dataset, outperforming previous models, and was able to generate complex SQL queries with high accuracy and fluency. Its accuracy was 68.0%, which at the time of publishing made it to the top places in the WikiSQL dataset leaderboard.

2.3. Microsoft's HydraNet model

The top three models on the WikiSQL leaderboard include HydraNet [3], X-SQL [1], SQLova [2], which all share a similar architecture. We will focus on the HydraNet arhitecture in this section.

This is multilayer arhitecture, which manages to obtain a 92.2% test execution accuracy on the WikiSQL dataset. The first layer encodes the input using the RoBERTa encoder. RoBERTa builds on BERT's language masking strategy and modifies key hyperparameters in BERT, including removing BERT's next-sentence pretraining objective, and

training with much larger mini-batches and learning rates. RoBERTa was also trained on an order of magnitude more data than BERT, for a longer amount of time. This allows RoBERTa representations to generalize even better to downstream tasks compared to BERT. The input sentences pair will be further tokenized and encoded to form the inputs of Transformer-based model. The second layer is supposed to strengthen the encoder output with bi-LSTM or self-attention. The goal of this step is to capture a larger context of an input. Bi-LSTM or self-attention enables the model to access information about prior, current, and future input. The resulting embeddings don't merely capture the local context of a word — they can, in principle, capture the entire semantic of the input, which is one of the arhitecture's advantages. The final layer consists of classification modules over six different components. Each module has its own attention weight to align the hidden representation and the final label.

Some of the disadvantages of this arhitecture include, again, the fact that the model doesn't work for join queries and the fact that the model might be computationally expensive to train and deploy.

2.4. Usage of transfer learning for text-to-SQL

Transfer learning, where a model is first pre-trained on a data-rich task before being fine-tuned on a downstream task, has emerged as a powerful technique in natural language processing (NLP). The effectiveness of transfer learning has given rise to a diversity of approaches, methodology, and practices.

One other approach for the text-to-SQL task is to try to use transfer learning on a large general usage language model, such as Google's T5 model. This is the approach we also want to explore. T5 is trained on a diverse set of texts, with a massive amount of data, which allows it to learn general-purpose representations that can be adapted to a wide range of tasks, such as text-to-SQL, which could represent an advantage for our goal. One possible disadvantage of this approach is that the fine tuned model might not adapt as well as a specially trained model on our task.

3. Proposed solution

Our proposed solution is to use Google's T5 model and apply transfer learning to it for the text-to-SQL task, using the WikiSQL dataset.

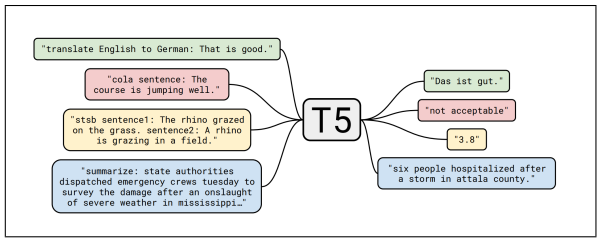

Figure 1. A diagram of our T5's text-to-text framework. Every task we consider (including translation, question answering, and classification) is casted as feeding the model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks.

The T5 model is based on the transformer architecture, which is an attention-based neural network that can handle long-term dependencies and large amounts of input data. T5 has been trained on a diverse set of texts, which allows it to learn general-purpose representations that can be adapted to different tasks. T5 has been shown to achieve state-of-the-art performance on several natural language understanding tasks and it's widely used.

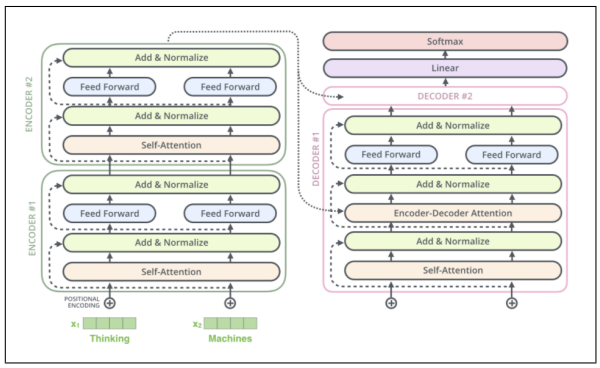

Figure 2. Google T5 model structure

In order to use the WikiSQL dataset with the T5 model, we need to preprocess the data to match the format expected by the used model. Next, we use a tokenization module for separating the input into smaller units called tokens. This input is further passed to our new model that we train based on T5. We experiment with the hyperparameters we are using and with the training arguments hoping to improve the model's performance. We try to use different batch sizes (e.g., 16, 32, etc.), epochs number (5 or 10) and different optimizers (e.g., Adam, AdamW) and compare the results we obtain. For evaluating, we use the Rouge precision, recall and Fmeasure metrics. ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The trained model is finally evaluated after the

training phase finishes, and we use it with a module which accepts a plain English query and attempts to convert it to a SQL query, using our new model.

The overview of our model can be followed in the diagram from below.

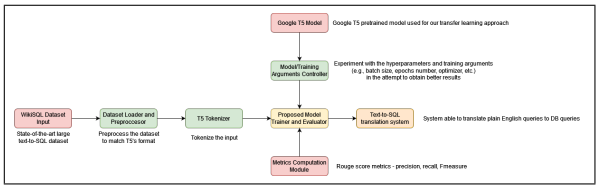

Figure 3. Model overview

4. Experimental results

For all of our experiments the dataset we use is WikiSQL and the underlying model used for transfer learning is Google T5.

4.1. First training scenario

Our first experiment used the following hyperparameters and training arguments:

- Optimizer: Adam Optimizer;
- Loss function: the cross-entropy loss function. Cross-entropy loss is commonly used in natural language processing tasks such as language modeling and text classification. It measures the difference between the predicted probability distribution and the true distribution of the target sequence;
- Learning rate: 0.00002;
- Weight decay: 0.01;
- Number of epochs: 5;
- Batch size: 16;

The training and validation loss we obtained for each training epoch are present in the following table:

Epoch	Training Loss	Validation Loss
1.0	0.2038	0.1658
2.0	0.1799	0.1495
3.0	0.1672	0.1401
4.0	0.1587	0.1333
5.0	0.1496	0.1286

Table 1. Training and validation loss

The Rouge metrics we obtained for each training epoch are present in the following table:

Epoch	Rouge2 Precision	R2 Recall	R2 Fmeasure
1.0	0.7861	0.6971	0.7318
2.0	0.8001	0.7086	0.7445
3.0	0.8066	0.7154	0.7513
4.0	0.8113	0.7196	0.7557
5.0	0.8145	0.7225	0.7588

Table 2. Rouge metrics

We can see that our model has an acceptable precision, compared to the existing models. While there are today state-of-the-art models that perform better, our model still outperforms some of the state-of-the-art models from a few years ago, which might still be a promising result. A lower recall as in our results suggests that the model is not correctly identifying as many of the positive instances as it should. This could mean that the model is misclassifying some of the positive instances as negative, or that it is failing to identify some positive instances altogether.

4.2. Second training scenario

Our second experiment used the following hyperparameters and training arguments:

- Optimizer: AdamW Optimizer;
- Loss function: the cross-entropy loss function;
- Learning rate: 0.00002;
- Weight decay: 0.01;
- Number of epochs: 5;
- Batch size: 16;

The training and validation loss we obtained for each training epoch are present in the following table. We can see that the results we obtained are very similar to the ones in the previous experiment, so all the observations stay the same.

Epoch	Training Loss	Validation Loss
1.0	0.2041	0.1665
2.0	0.1843	0.1515
3.0	0.1710	0.1451
4.0	0.1591	0.1383
5.0	0.1501	0.1298

Table 3. Training and validation loss

The Rouge metrics we obtained for each training epoch are present in the following table:

Epoch	Rouge2 Precision	R2 Recall	R2 Fmeasure
1.0	0.7791	0.6931	0.7278
2.0	0.7951	0.6996	0.7315
3.0	0.8016	0.7024	0.7401
4.0	0.8043	0.7086	0.7522
5.0	0.8102	0.7135	0.7543

Table 4. Rouge metrics

4.3. Third training scenario

The main difference compared to the previous experiments in that we changed the epochs number to 10 and the batch size to 32. This slightly improved the performance of our model, which might suggest that further increasing the epochs number might help, but the difference seems to not be a major one. Also, this increases the training time and the complexity of our model. The used hyperparameters and training arguments are the following ones:

- Optimizer: AdamW Optimizer;
- Loss function: the cross-entropy loss function;
- Learning rate: 0.00002;
- Weight decay: 0.01;
- Number of epochs: 5;
- Batch size: 32;

The training and validation loss we obtained for each training epoch are present in the following table:

Epoch	Training Loss	Validation Loss
1.0	0.2718	0.210285
2.0	0.2172	0.176214
3.0	0.1982	0.160817
4.0	0.1830	0.150414
5.0	0.1702	0.143255
6.0	0.1631	0.137783
7.0	0.1575	0.133583
8.0	0.1520	0.129966
8.0	0.1458	0.126640
10.0	0.1422	0.124164

Table 5. Training and validation loss

The Rouge metrics we obtained for each training epoch are present in the following table:

Epoch	Rouge2 Precision	R2 Recall	R2 Fmeasure
1.0	0.7551	0.6695	0.7026
2.0	0.7779	0.6893	0.7237
3.0	0.7918	0.7014	0.7369
4.0	0.8006	0.7100	0.7456
5.0	0.8052	0.7137	0.7497
6.0	0.8086	0.7166	0.7529
7.0	0.8123	0.7203	0.7566
8.0	0.8154	0.7234	0.7597
9.0	0.8171	0.7251	0.7613
10.0	0.8193	0.7267	0.7631

Table 6. Rouge metrics

4.4. Example model output

An example of the output of the model on several user queries expressed in natural language can be seen below:

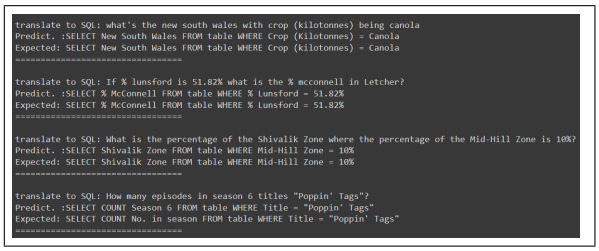

Figure 4. Model example output

We notice that it confirms the metrics we obtained during training and testing. The model generally performs well, especially on queries that are not too complex.

5. Conclusion

In conclusion, this paper explored the potential of using transfer learning with the Google T5 model for text-to-SQL tasks. Our experiments demonstrated that fine-tuning a pre-trained T5 model on a text-to-SQL dataset can achieve an acceptable performance on these tasks. These findings suggest that transfer learning with T5 may be a promising approach for text-to-SQL tasks and could be further explored in future research. Additionally, the T5 model may be a useful tool for other text-to-structured data tasks as well.

References

[1] Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*, 2019. 2

[2] Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql

with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*, 2019. 2

[3] Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-sql. *arXiv preprint arXiv:2008.04759*, 2020. 2

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020. 1

[5] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017. 2

[6] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle 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. 2

[7] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017. 2