Byte-Pair Encoding for Text-to-SQL Generation

Samuel Müller
University of Cambridge
samuelgabrielmuller@gmail.com

Andreas Vlachos
University of Cambridge
av308@cam.ac.uk

Abstract

Neural sequence-to-sequence models provide a competitive approach to the task of mapping a question in natural language to an SQL query, also referred to as text-to-SQL generation. The Byte-Pair Encoding algorithm (BPE) has previously been used to improve machine translation (MT) between natural languages. In this work, we adapt BPE for text-to-SQL generation. As the datasets for this task are rather small compared to MT, we present a novel stopping criterion that prevents overfitting the BPE encoding to the training set. Additionally, we present AST BPE, which is a version of BPE that uses the Abstract Syntax Tree (AST) of the SQL statement to guide BPE merges and therefore produce BPE encodings that generalize better. We improved the accuracy of a strong attentive seq2seq baseline on five out of six English text-to-SQL tasks while reducing training time by more than 50% on four of them due to the shortened targets. Finally, on two of these tasks we exceeded previously reported accuracies. The implementation is available at https://github.com/SamuelGabriel/sqlbpe.

1 Introduction

Information is often stored in relational databases, but these databases can only be accessed with specialized programming languages, like the Structured Query Language (SQL). A natural language interface to a database (NLIDB) is a system that allows users to ask the database questions or give commands in natural language instead of using any programming language e.g. by mapping natural language to SQL queries like in the example in Figure 1. Advances in text-to-SQL generation can also be relevant to other tasks concerned with the generation of other programming languages from natural language.

Earlier approaches to solve this task were

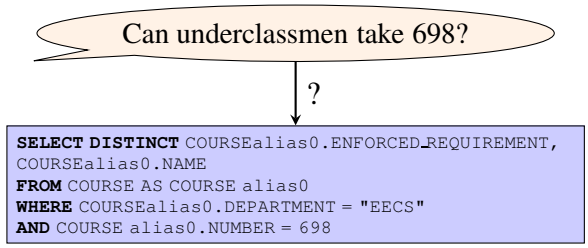

Figure 1: The task of text-to-SQL generation is to generate a query given a question in natural language. This is an example from the Advising dataset (Finegan-Dollak et al., 2018).

either rule-based (Popescu et al., 2004) or based on statistical machine translation models (Andreas et al., 2013). Recently, Finegan-Dollak et al. (2018) showed strong results using neural sequence-to-sequence models on English-language text-to-SQL task, in line with other recent work by (Iyer et al., 2017; Dong and Lapata, 2016), even though these models have to predict rather long sequences when predicting SQL queries. For comparison, the translation dataset Multi30k (Elliott et al., 2016) for example has a mean target length that is less than a sixth of that of targets in the Advsising (Finegan-Dollak et al., 2018) dataset, one of the text-to-SQL datasets we evaluate on. The average length of the targets in the Multi30k dataset increases to over three quarters of the average length in Advsising when predicting on the character-level. Interpreting the sentence as a sequence of characters gives the model more flexibility and allows it to predict previously unseen words, but the increased sequence length also takes its toll on predictive accuracy and speed (Sennrich et al., 2016). As a remedy, Sennrich et al. (2016) proposed to apply the Byte-Pair Encoding algorithm (BPE) (Gage, 1994)

on target sequences for character-level machine translation. BPE is a compression algorithm that encodes commonly co-occurring characters into single symbols.

In this work we use BPE to compress token-level SQL targets to shorter sequences with a flexible stopping criterion and guidance by the abstract syntax tree (AST). We improved the accuracy of a strong attentive seq2seq baseline on five out of six English text-to-SQL tasks while reducing training time by more than 50% on four of them due to the shortened targets. Finally, on two of these tasks we exceeded previously reported accuracies. We are able to improve on strong baselines in accuracy, training time and inference time with our methods on most SQL-to-text tasks.

2 Background

In the text-to-SQL task we want to learn a model p of SQL queries of the form $a = y_1 \dots y_{|a|}$ conditioned on natural language input of the form $q = x_1 \dots x_{|q|}$. We want to find a p such that our accuracy is high on the test set. We approximate this goal by search for a p from some class of models that is as close as possible to $\arg\max_p \prod_{(q,a)\in T} p(a|q)$, where T is the test set. For this purpose we use the sequence-to-sequence neural network approach (Sutskever et al., 2014) which constructs the SQL query incrementally, i.e. $p(a|q) = \prod_{t=1}^{|a|} p(y_t|y_{1:t-1}, q)$. In this framework the question q is encoded and decoded with a Long Short-Term Memory unit (LSTM) Hochreiter and Schmidhuber (1997). Additionally we employ an attention mechanism (Bahdanau et al., 2014) to allow the decoder to flexibly combine the states of the encoder.

3 BPE for text to SQL generation

While Sennrich et al. (2016) applied BPE (Gage, 1994) on characters inside words, we apply BPE on tokens, since the vocabulary of SQL queries is quite restricted (ignoring named entities); e.g. the anonymized Advsising dataset contains only 177 unique tokens. BPE is a compression algorithm that groups commonly co-occurring symbols into single symbols. BPE works by iterating over a dataset in k scans, where k is given. In each scan, the most frequent pair of consecutive symbols is replaced with a new symbol. This way, frequent co-occurring sequences of symbols can be predicted in one step, simplifying the task. In this paper, we apply BPE to tokenized queries by interpreting each token as a symbol and combining neighboring tokens as pairs to create new tokens.

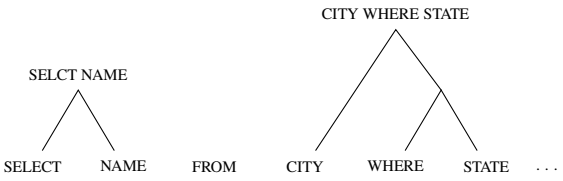

Figure 2: This figure shows the encoding of an example query in part (a) and the applied BPE rules in part (b).

Figure 2 illustrates an application of BPE encoding, where each node represents one token. Throughout the paper we use the training and validation sets to build and tune the BPE encoding on to ensure that the models developed are fairly assessed on the test sets. To encode a new dataset with a given list l of BPE encoded entries, one follows the same procedure as for the generation of the BPE encoding, but just applying rules found previously instead of creating new ones.

Algorithm 1: BPE with stopping criterion

Input: A training set $\mathbf{D}$ of target queries, and a validation set $\mathbf{V}$ of target queries. The number of retention steps r and the minimum number of occurrences in the training set for tokens appearing in the validation set m.
Output: A list l of pairs of tokens representing a BPE encoding as well as BPE encoded versions of $\mathbf{D}$ and $\mathbf{V}$.

```
l = EmptyList();
while diff < r and D contains a bigram do
    maxpair = pairWithMaxCount(D);  // maxpair is the most common token pair in D.
    newoov = addsNewOOV(D, V, maxpair, m);
    if newoov then
        diff += 1;
    else
        l.append(maxpair);
        D, V = replacePair(D, V, maxpair);
          // Replace occurences of maxpair with a new token.
end
Procedure addsNewOOV (D, V, pair, c)
    oovcount = |vocabulary(V, 1)\vocabulary(D, c)|;
      // vocabulary returns the set of tokens with a minimum number of occurences in a dataset.
    D̄, V̄ = replacePair(D, V, pair);
    newoovcount = |vocabulary(V̄, 1)\vocabulary(D̄, c)|;
    return newoovcount > oovcount;
```

4 A stopping criterion for BPE

In previous work the number of BPE scans k was treated as a hyper-parameter that needs to be hand tuned (Sennrich et al., 2016). While this might work for tasks where the performance of the model is robust against different values of k, we found that this is not the case for text-to-SQL generation, due to the small size of the datasets. If k is set too low, we do not experience all the benefits of BPE, since we could shorten sequences further. If k on the other hand is set too high, there is a risk that our model is unable to predict some combinations of tokens. This situation might arise, for example, if in the training set a token a is always followed by a token b, and therefore the model combines these two tokens into a new token x. Since all occurrences of a are followed by b, applying a BPE step will remove a completely from the dataset. Therefore if there is a sequence in the test data that requires generating a without b following it the model would not be able to.

To ameliorate this issue we propose a stopping criterion for BPE as outlined in Algorithm 1, which has two less sensitive hyper-parameters, r and m, instead of the number of steps k. We were able to use the same settings for these new parameters on many different datasets and tasks with competitive results, which preliminary experiments showed not to be possible with a fixed number of steps k. The method is outlined in algorithm 1. We keep track of all tokens present in the training and the validation set as we apply consecutive BPE steps and stop as soon as we took r steps that leave tokens in the validation set that can't be found in the training set no more. The second parameter m is the minimum number of occurrences in the training set for each token in the validation set. This is equivalent to ensuring that a minimum count is fulfilled for each token added to the vocabulary with BPE.

5 AST BPE

It was previously shown that it can be helpful to consider the abstract syntax tree (AST) of SQL queries for query generation (Dong and Lapata, 2018). Similar to the AST, the BPE algorithm defines a tree structure onto queries, but it might not be well aligned with the query's AST.

The idea of AST BPE is to keep the main principle of BPE, but align the BPE structure with the AST by restricting what is interpreted as a pair when computing a BPE encoding to subsequences that are aligned in the AST. More formally, consider two tokens a and b, that represent the token sequences $a_1, \ldots, a_{|a|}$ and $b_1, \ldots, b_{|b|}$ respectively, in the dataset $\mathbf{D}$ after a number of BPE steps. In the AST BPE setup the PairCounter in Algorithm 1 only considers a and b as neighbors if the sequence built by concatenating the two represents a set of neighboring sibling AST nodes. An example query encoded with AST BPE can be found in Figure 3. The colored boxes illustrate the levels of the query's AST. This method is especially helpful for the small datasets found in the text-to-SQL domain, since on larger ones that represent the target distribution well, vanilla BPE is likely to chooses tokens in a way such that are aligned with the AST. On small datasets like GeoQuery on the other hand, we could see that vanilla BPE for example encodes closing parentheses and following keywords as BPE tokens.

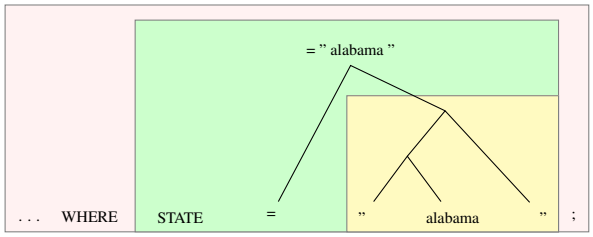

Figure 3: Illustration of an example query's AST BPE representation. Boxes of different color illustrate different levels in the AST.

6 Related Work

As far as we know there is no previous work on the application of BPE to the text-to-SQL task or any other structured language generation task. There exists research on related topics though. Dong and Lapata (2018) explored how to use a flexible form of templates by using a neural sequence-to-sequence model which generates targets that only contain a coarse representation of the query that is filled with entities and identifiers by a second model. In contrast to BPE this model increases the complexity of the approach as two models need to be trained. Finegan-Dollak et al. (2018) proposed a baseline model that is only based on templates, which are not dynamically predicted but gathered from the training data. Based on the representations of

queries in the anonymized version of a dataset, they built an index of distinct queries (up to entity names). Then, they used an LSTM-based classifier to classify a question as one of the anonymized queries in the training set and classify input tokens as entities that replace placeholders. This technique only yields a simple baseline, since it does not generalize to unseen queries. We refer to this method as the FD&K baseline. The AST of SQL queries was previously used to predict queries by Finegan-Dollak et al. (2018), who applied the methods Dong and Lapata (2016) developed for logical parsing to SQL. They generate the query recursively over the AST, while we use a sequential representation of the query at prediction time and just use tree structures over the queries to find common parts of queries to unify. We refer to the adaptation of this method to SQL by Finegan-Dollak et al. (2018) as D&L seq2tree. Iyer et al. (2017), similar to Jia and Liang (2016) used the tree structure inherent to logical forms and artificial languages to grow their datasets and therefore also implicitly teach the model the modularity and replaceability of nodes in the parse tree. In the following we will call this method Iyer et al.

7 Results

We report our results for both BPE and AST BPE with an attention-enabled seq2seq model for all datasets and report the accuracy of predicting the whole query exactly as the target query. We evaluate on the Advsising dataset, which counts 4570 questions, as well as the simultaneously re-published datasets ATIS (Finegan-Dollak et al., 2018; Deborah A. Dahl and Shriber, 1994; Srinivasan Iyer and Zettlemoyer, 2017), an air traffic related dataset of 5280 questions, and GeoQuery (Finegan-Dollak et al., 2018; Zelle and Mooney, 1996; Srinivasan Iyer and Zettlemoyer, 2017), a dataset regarding the geography of the United States of America of 877 questions. All datasets have English as their natural language part.

Our evaluation encompasses experiments on two split; one where the datasets in train, validation and test set are split based on the questions asked, which have examples with the same target across these sets, and one split based on the query, where each query is only contained in one of the sets. We used the same hyper-parameters for both dataset splits. For all experiments the retention steps parameter r was set to 20 and the minimum frequency in the training set m was set to 100 for all datasets, besides Advsising for which it was set to 300. We ran all experiments on a single Nvidia Tesla M60 GPU. In table 1 our results on the test sets can be found.

Training Details All models use bidirectional LSTMs for encoding, with a hidden state size of 100. We initialize the LSTMs for encoding with zeroed hidden states. For decoding we use a LSTM and treat the initial hidden state as a parameter. We used concurrently-trained token embeddings of size 100 for all models. To extract the AST from the queries in the training set for AST BPE we use the Python library 'sqlparse' by Andi Albrecht.[1]

During training we applied a dropout of 0.5 on the input and output of the LSTMs. We used batches of size 32 and the Adam optimizer (Kingma and Ba, 2014) with a constant learning rate of 1e-3. All weights were initialized with the default PyTorch weight initialization (Paszke et al., 2017). At inference time beam search with a beam width of 3 was applied. We employed early stopping guided by the validation set and a retention period of 50 epochs.

Evaluation on the question split On the question splits BPE outperforms the attentive sequence-to-sequence for both ATIS and GeoQuery in both accuracy and training time, while especially Iyer et al. could gain further improvements. These could be combined with both versions of BPE though. On GeoQuery, the AST BPE model outperforms the attentive sequence-to-sequence model by over 2% in absolute accuracy, establishing a new state-of-the-art, and requiring only about a third of the time to train.

Advsising is the only dataset on which we could not improve performance with BPE. Since Advsising 's question split is structured such that each query in the question split of Advsising appears in the training, the test and the validation set, a BPE encoding with a minimum count m of 1 and 1 retention step r can reduce the whole task to a classification task. BPE with these setting achieves an accuracy of 90.40% with a training time of 48 minutes and a inference time of 15 seconds. It is worth noting that this setup surpasses even the FD&K

[1] https://github.com/andialbrecht/sqlparse

Model	BPE Usage	Question Split Advsising	Question Split ATIS	Question Split GeoQuery	Query Split Advsising	Query Split ATIS	Query Split GeoQuery
Seq2seq	No BPE	79.93% 9h12m 3m43s	51.68% 28h20m 5m47s	59.14% 1h11m 1m47s	0.00% 3h43m 8m38s	8.65% 6h31m 4m44s	0.00% 1h58m 0m25s
Seq2seq with attention	No BPE	**89.01%** 6h8m 4m36s	56.60% 22h40m 6m29s	70.25% 1h36m 1m1s	3.66% 4h27m 11m38s	25.94% 11h16m 5m51s	47.25% 2h43m 1m36s
	BPE	88.13% 3h56m 2m18s	57.05% 11h51m 4m16s	70.61% 24m18s 1m45s	3.06% 3h45m 6m24s	6.05% 2h41m 2m36s	39.01% 1h21m 0m24s
	AST BPE	87.61% 3h19m 3m34s	56.38% 12h31m 4m25s	**72.40%** 31m5s 1m46s	3.71% 5h1m 6m28s	24.50% 4h35m 3m43s	**50.00%** 1h24m 0m28s
FD&K baseline	No BPE	89%	56%	56%	0%	0%	0%
D&L seq2tree	No BPE	88%	56%	68%	**8%**	**34%**	23%
Iyer et al.	No BPE	88%	**58%**	71%	6%	32%	49%

Table 1: Accuracy alongside training (the upper time) and testing time (the lower time) on the datasets. Accuracy is measured on the test set. The results below the line are from Finegan-Dollak et al. (2018).

baseline, although the only difference to it is that our model has an attention mechanism, for a simple classification.

To further investigate the reasons for the good results with BPE methods across the datasets we took a closer look at the performance of AST BPE on GeoQuery for unseen and seen queries. A query is *seen*, if it is contained in the training data with a different question. Table 2 shows that for the GeoQuery task AST BPE did actually not improve accuracy on unseen queries, but instead improved the accuracy on seen queries, which make up 77% of the test set.

Query type	Seen queries	Unseen queries
No BPE	84.26%	22.22%
AST BPE	88.43%	17.46%

Table 2: Accuracy on GeoQuery for queries (not) in the training set.

Evaluation on the query split We could improve performance on the query splits of both Advsising and GeoQuery with AST BPE. For GeoQuery we set a new state-of-the-art on this task, and at the same time halve inference and test time compared to the base model. For Advsising, the training time did not improve, even though AST BPE reduces the average query length in the training set by over 44%; the effect of this could be seen in improved inference speed.

Only for ATIS the BPE models did not improve accuracy, but training time could be more than halved at an absolute accuracy loss of less than 1.5% with AST BPE. A likely reason for why BPE did not improve accuracy on ATIS, is that ATIS contains many different query patterns, a pattern being a query type abstracted away from the table schema. Each pattern in ATIS only appears in 7 queries on average. In the test set of ATIS' query split over 47.84% of the queries therefore have an unseen pattern, while on the query split of GeoQuery for example only 5.49% of queries have an unseen pattern.

BPE setting	Unseen Patterns	Seen Patterns
No BPE	25.90%	23.20%
BPE	1.20%	11.05%
AST BPE	24.70%	25.41%

Table 3: This table shows the accuracy of seq2seq models with attention and different kinds of BPE on the queries from the test set with patterns (not) contained in the dataset. It can be seen that BPE and AST BPE are especially strong in predicting queries with a known pattern.

In Table 3 we analyze the accuracy of different models on the queries with seen and unseen patterns on ATIS. We can see that for BPE, which performs overall worst on this dataset, the performance is even worse for unseen patterns. For AST BPE the outcome is somewhat similar, as it improves performance on seen patterns, but the performance on unseen patterns degrades. This result aligns well with what we saw in Table 2 for AST BPE on the question split of GeoQuery, where we could see that AST BPE helped with seen queries, but not with unseen ones.

8 Conclusion

In this work we showed that BPE can be applied to text-to-SQL generation. In particular we found that for anonymized datasets BPE was able to improve upon the base models in five out of six cases, and additionally cut the training time by more than 50% in four of the cases. We showed that AST BPE is especially helpful for datasets split by query, which require the model to generalize to previously unseen queries and query structures. We could also observe that the biggest impact was on experiments with the smallest dataset,

GeoQuery, where we achieved new state-of-the-art results for both dataset splits. The BPE methods developed in this work are not specific to SQL and could be applied to many other tasks requiring structured language generation.

Acknowledgments

Andreas Vlachos is supported by the EPSRC grant eNeMILP (EP/R021643/1).

References

Jacob Andreas, Andreas Vlachos, and Stephen D. Clark. 2013. Semantic parsing as machine translation. In *ACL*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Michael Brown William Fisher Kate Hunicke-Smith David Pallett Christine Pao Alexander Rudnicky Deborah A. Dahl, Madeleine Bates and Elizabeth Shriber. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. *Proceedings of the workshop on Human Language Technology*, pages 43–48.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742.

D. Elliott, S. Frank, K. Sima'an, and L. Specia. 2016. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 351–360.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal*, 12:23–38.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistic In *Proceedings of Coling 2004*, pages 141–147, Geneva, Switzerland. COLING.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Alvin Cheung Jayant Krishnamurthy Srinivasan Iyer, Ioannis Konstas and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programmin In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055.