Context, dimensions, ensembles: TypeSQL with BERT

Lukas Muttenthaler
University of Copenhagen
IT and Cognition
mnd926@alumni.ku.dk

Abstract

To query a database, questions expressed in natural language are required to be mapped into a well-formed logical representation that computers can easily read. Such a well-formed language is SQL. In this study, we focus on the approach of translation natural language to SQL, and experiment with an existing architecture that has proven to be successful, namely TypeSQL. The aim of the study was to investigate which features and word representations can help to query a database. We replace context-free GloVe embeddings with contextualized BERT embeddings, compare lower dimensional vectors with high dimensional vectors, and test ensembles against single models. Ablation studies show that BERT models do not add any benefits - at least not in the way we have deployed BERT -, lower dimensional vectors are not notably inferior to higher dimensional vectors, and ensembles outperform single models only on the SELECT clause.

1 Introduction

An enormous amount of the world's data is stored in relational databases. Accessing this data requires the user to be adept at query languages such as SQL. This can be a daunting task for someone without a technical background. Hence, to facilitate the users' lives and automatically query a database, it is essential to develop algorithms that are capable of accurately mapping natural language questions into well-formed logical representations. This endeavour is a semantic parsing[1] problem in its nature.

One task in the vast field of semantic parsing is the Text-to-SQL problem, or in short NL2SQL[2]. It has recently attracted wide interest in industry (Zhong et al., 2017) but has been a long standing problem in academia in both the Computational Linguistics (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Xu et al., 2017; Yu et al., 2018) and Database community (Li and Jagadish, 2014) which approach the problem from different angles. Here, we solely focus on the Natural Language Processing side of the problem. Since neural network approaches have recently become widely used in NLP due to their promising results, they will be the focal point of this study.

Contrary to previous work, we exploit BERT embeddings (Devlin et al., 2018) to approach the NL2SQL problem. BERT refers to Bidirectional Encoder Representations from Transformers. Historically, the objective of language models was to predict words given the right or left context of neighbouring words of some window size (e.g., previous *n*-gram) (Ratnaparkhi, 1997). Until the advent of BERT, all language models were deployed unidirectional or shallowly bidirectional (Peters et al., 2017, 2018), and as such not capable of contextualizing a word given the entire context the word appears in. BERT, however, has closed the gap and, as the name suggests, exploits the context both to the left and right of a word. BERT is the first unsupervised, deeply bidirectional language model for NLP. As such, BERT became indispensable in the disentanglement of a word's context on a variety of NLP tasks, as numerous recent studies, conducted in both academia and industry, have shown (Cheng et al., 2019; Glass et al., 2019; Zhang et al., 2019), and the GLUE leaderboard indicates (Wang et al., 2018a), where models that deploy BERT, or optimised versions of BERT (Liu et al., 2019), outperform more traditional approaches.

2 Related Work

The first sequence-to-sequence neural model that significantly outperformed both existing neural se-

[1] Semantic parsing is the task of converting a natural language utterance into a machine readable representation of its meaning.

[2] Here, NL is the abbreviation for Natural Language.

mantic parsers (Dong and Lapata, 2016) and more traditional rule-based models (Liang et al., 2013; Zhao and Huang, 2014) was Seq2SQL (Zhong et al., 2017). Their model pushed the state-of-the-art semantic parsing model (Dong and Lapata, 2016) from $\sim$ 36% to almost 60%, and, for the first time, showed that the long standing NL2SQL problem might be solvable. Zhong et al., 2017 exploited a deep sequence-to-sequence neural network using reinforcement learning (RL). To train their model, they used a mixed objective, consisting of minimising cross entropy losses and maximising RL rewards. In additon to their model, they released WikiSQL (Zhong et al., 2017), the currently biggest crowd-sourced dataset with $80,654$ pairs of questions and corresponding, hand-annotated SQL queries. WikiSQL[3] serves as the benchmark dataset, which is utilized to compare different architectures against each other. One important assumption of the WikiSQL task is that each token in the output SQL query is either a column name of the SQL table or a sub-string of the natural language question, coupled with an SQL keyword (see Table 1). This is crucial when the model is not provided with the database table content.

- Question: How many players from York College are position DB?
- SQL: **SELECT** COUNT Players **WHERE** (College = "York") **AND** (Position = "DB")

Index	CFL Team	Player	Position	College
...	...	...	...	...
27	Hamilton Tiger-Cats	Connor Healy	DB	Wilfrid Laurier
28	Hamilton Tiger-Cats	Anthony Forgone	DB	York
29	Ottawa Renegades	L.P. Ladouceur	DT	California
30	Toronto Argonauts	Frank Hoffman	DL	York
...	...	...	...	...

Table 1: WikiSQL example. Inputs consist of a table and a natural language question (see above). Outputs consist of a ground truth SQL query and the corresponding execution result.

Since the release of WikiSQL, many followed Zhong et al., and developed models that achieved state-of-the-art performances (Xu et al., 2017; Guo and Gao, 2017; Wang et al., 2018b; Yu et al., 2018), and outperformed traditional semantic parsers which are not aware of the output grammar. This awareness of the SQL grammar, which is achieved through a sketch-based approach, makes it easier for existing models to make accurate predictions. The sketch highly resembles the SQL schema, and as such, contains a dependency graph. This enables a model to base a prediction only on the n previous predictions it depends on. Thus, instead of solving two tasks, generating the output grammar and predicting the values of the different clauses, current models only need to focus on the latter task.

The first model, which completely avoided the sequence-to-sequence structure which suffers from the "order-matters" problem, however, was SQLNet (Xu et al., 2017). This is reasonable, when the order does not matter - as is the case with columns in a schema table. Their objective was to predict an unordered set of constraints instead of an ordered sequence, which changed the encoder-decoder nature from a sequence-to-sequence to a sequence-to-set model. Instead of generating a sequence of column names, only a subset is predicted. TypeSQL (Yu et al., 2018) developed this approach further, and added in addition to the sequence-to-set objective and the column attention mechanism, employed by SQLNet (Xu et al., 2017), type embeddings. Words were represented through GloVe (Pennington et al., 2014), and type embeddings were self-trained via an embedding layer. For each word, type embeddings were then concatenated with GloVe before fed into the first recurrent hidden layer of a bi-directional Long-Short-Term-Memory network. We will explain the model in more detail in the following section.

3 Method

We resume TypeSQL's approach, and test different versions of their architecture, which we have modified accordingly. Since we have exploited TypeSQL as the foundation for all of our models, most of the code is based on TypeSQL [4] (Yu et al., 2018), and hence on SQLNet [5] (Xu et al., 2017). Instead of utilizing reinforcement learning, where the decoder is rewarded for generating the correct query, we employ a sketch-based approach and perceive the NL2SQL problem as a slot filling task. The slots are arguments of the different clauses in the SQL query. The WikiSQL sketch looks as follows:

- **SELECT** $AGG $COL
- **WHERE** $COL $OP $VALUE

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

[4] https://github.com/taoyds/typesql/

[5] https://github.com/xiaojunxu/SQLNet/

- (**AND** $COL $OP $VALUE)*

SQL keywords are displayed in boldface, and slots in blue. During the generation of the SQL query, all slots with a preceding $ have to be filled. As predictions in the WikiSQL task have to be made with respect to one table only, the usually common FROM clause can be bypassed. In the SELECT clause only a single column is predicted. For the WHERE clause, however, one or more columns are filled, depending on the number of additional AND clauses the model predicts. The Kleene star operator (*) indicates that any query can consist of zero or more AND clauses. $COL has to be filled with a column name of the SQL table, $OP can take any of the following operators ($<, =, >, \geq, \leq$), and $VALUE must be a sub-string of the natural language question (if the database table content is not provided).

The architecture of our model is highly similar to TypeSQL (Yu et al., 2018), which served as the foundation of our approach. Hence, the task at hand is not a sequence-to-sequence but rather a sequence-to-set task, which was first employed in SQLNet (Xu et al., 2017). That is, goal of the query generation is to predict an unordered set of constraints instead of an ordered sequence.

3.1 BERT

The aim of the study was to compare the performance of contextual BERT embeddings against context-free GloVe within the architecture of TypeSQL. Hence, we deployed the pre-trained, uncased BERT$_{BASE}$ model with 12 layers, 768 hidden units, 12 heads, and 110M parameters. Uncased means that the text has been lower-cased before WordPiece tokenization. We employed the pre-trained base version that is publicly available on the PyTorch-Transformers GitHub repository [6].

3.2 Retokenization

Since all natural language questions and the corresponding SQL queries in the WikiSQL dataset are split into word tokens, one is required to either rejoin sub-word or character tokens into full words or tokenize the natural language questions according to BERT's WordPiece model, and rewrite the SQL queries accordingly. We did not pursue the latter approach due to two reasons.

First, owing to simplicity and time constraints, it was faster, and thus more convenient for us to program a string-matching algorithm that rejoins sub-words and characters into full words, and merge the contextual embeddings accordingly.

Second, since we wanted to employ GloVe embeddings for both the prediction of the aggregate value in the SELECT clause, and the representation of column names in the SQL table, we had to keep a full-word token representations of each natural language question. The full version of our string-matching algorithm can be downloaded from our GitHub repository and works as depicted in Algorithm 1.[7]

Algorithm 1: Retokenizer

Input : WordPiece tokens; WikiSQL tokens
Output: Rejoined tokens

```
j = 0;
for i, WP tok ∈ WordPiece toks do
    for tok ← toks[j] ∈ WikiSQL toks do
        if tokens match then
            j ← j + 1;
            break;
        else
            k ← WikiSQL toks[j+1];
            ∀ WP toks[i:k] ≠ tok ← Rejoin;
            j ← j + 1;
            break;
```

We suppose this algorithm could be useful for a number of other tasks where the goal is to rejoin sub-word or character tokens into full-word tokens, and merge the corresponding word vector representations. The latter step is not compulsory, and can easily be omitted. Moreover, we acknowledge the fact that the latter will potentially lead to information loss. We tested three different merging strategies, although report results only for two of them. We either averaged, max-pooled or summed the contextual embeddings across all sub-word tokens that had to be rejoined into full-word tokens. For max-pooling we only kept the max value per dimension. All merging strategies were performed over the feature values of the respective word embeddings (i.e., column-wise).

During initial experiments on a toy version of the dataset, we figured that summing across the

[6] https://github.com/huggingface/pytorch-transformers

[7] https://github.com/LukasMut/TypeSQL-plus-BERT

features leads to significantly poorer performance than averaging or max-pooling. Thus, we have not further pursued this strategy.

We had to perform accent stripping, and uni code normalization due to unequal character reprentations between WordPiece and WikiSQL tokens. Due to noise in the data (e.g., questions that consisted of both Tamil and English characters), we could, unfortunately, not retokenize all questions in the dataset. Furthermore, a significant number of Japanese, Chinese or Korean characters is represented as [UNK] tokens by the WordPiece model.

Hence, for those languages we performed a hack that will merge characters that the model is aware of (e.g., original Chinese character) with characters that are not part of the model's vocabulary (i.e., [UNK]). This is not clean, and most likely leads to representations that can be regarded as noise, but we did not see an alternative to this approach other than dropping such questions. That left us with a total of 99.4% of questions whose byte-pair encoded sub-word tokens could be rejoined into WikiSQL full-word tokens. Whereas we simply dropped questions that could not be retokenized in training, we counted his a miss for the model during evaluation on the test set.

3.3 Model

For GloVe models, GloVe (Pennington et al., 2014) and paraphrase embeddings of the same dimensionality were concatenated to represent words, whereas BERT models exploited BERT embeddings (Devlin et al., 2018) only. Hence, GloVe were either of dimensionality 50 or 300, whereas BERT embeddings had 100 or 600 dimensions. For each model, word embeddings were concatenated with self-trainable type embeddings of lower dimensionality before fed into the first hidden layer of the encoder.

Type embeddings were half of the size of word embeddings, that is 50 or 300 dimensions. That resulted in an input matrix of $|S| \times 150$ for small, and $|S| \times 900$ for large models per question, where $|S|$ denotes the length of the sentence (i.e., number of words). For how questions were tokenized to recognize word types, look at the implementation of TypeSQL (Yu et al., 2018).

3.3.1 Bi-directional LSTM

As common in modern NLP systems, the encoder of this model exploits Long-Short-Term-Memory cells (LSTMs) (Hochreiter and Schmidhuber, 1997). For each question, word and corresponding type embeddings are concatenated and fed into a bi-LSTM. That is, questions are encoded forward and backward in time. Hidden vectors are computed as follows.

$$\overrightarrow{\mathbf{h}}_{QT(t)} = \text{LSTM}\left(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t\right), t = 1, \cdots, |x| \quad (1)$$

$$\overleftarrow{\mathbf{h}}_{QT(t)} = \text{LSTM}\left(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t\right), t = |x|, \cdots, 1 \quad (2)$$

$$\mathbf{H}_{QT(t)} = \left[\overrightarrow{\mathbf{h}}_{QT(t)}, \overleftarrow{\mathbf{h}}_{QT(t)}\right] \quad (3)$$

LSTM denotes the LSTM function (Hochreiter and Schmidhuber, 1997), $\text{h}_{QT(t)}$[8] is the hidden state at time step t, and x represents the input question represented by concatenated word and type embeddings.

Firstly, the LSTM processes a question from left to right, that is forward in time. Secondly, the LSTM processes a question from right to left, that is backward in time. Lastly, the last hidden vectors of both processing mechanisms are concatenated to form a bi-directional representation of the words and types respectively, which results in H_{QT} as the final hidden state per bi-LSTM layer for each input question. A second bi-LSTM is run over the column names in the SQL table to produce the hidden state H_{COL}. This is done for each hidden layer in the encoder.

3.3.2 Slot-Filling

TypeSQL exploits three different models to generate the values for the different SQL clauses.

- Model_AGG for $AGG
- Model_COL for $COL_SEL, $COND_# and $COL_WHERE
- Model_OPVAL for $OP, and $COND_VAL

The parameters of bi-LSTM_{QT} and bi-LSTM_{COL} are shared in each model. Thus, there are 6 models in total.

According to (Yu et al., 2018) Model_{AGG} converges faster and suffers from overfitting when combined with other models. Thus, it was employed separately. We tried to predict $AGG with exploiting BERT embeddings, but observed significantly lower scores for $AGG with BERT than without. Hence, we utilized GloVe to predict

[8] QT denotes question and type embeddings.

$AGG throughout all models, no matter whether the model was based on GloVe or BERT. Since all models relied on GloVe, we could thus not exploit full-dimensional BERT embeddings, but were required to use the same dimensionality for both context-free and contextual embeddings.

3.3.3 Column attention

Column attention was first introduced in SQLNet (Xu et al., 2017), and therefore also used in TypeSQL (Yu et al., 2018). Attention is a mechanism that was first proposed by (Bahdanau et al., 2014), and further developed in (Luong et al., 2015) and (Vaswani et al., 2017). It is inspired by language processing mechanisms of the human brain during reading, and as such helps artificial neural networks to better understand natural language.

At most times, not all words but a few are important in a question to either generate a query or answer a question. Thus, attention aims at computing so-called attention scores that reflect which words of the sentence at the current time step are the most informative for the decoder to generate the corresponding output. The higher the attention score, the more informative the word.

The attention score is computed as the dot product between the last hidden state ${H^i}_{COL}$ of the bi-LSTM$^i{}_{COL}$ and the transpose of the dot product between the weights matrix W^i and the last hidden state ${H^i}_{QT}$ of bi-LSTM$^i{}_{QT}$

As a following step, weights are computed through deploying the softmax function over the attention scores, which reflect how much weight the model shall put on the respective word in the input question X_i. Again, the higher the attention weight, the more information lies in the corresponding word. As a final step, the dot product between attention weight vector α and the hidden state ${H^i}_{QT}$ is computed.

$$softmax = \frac{\exp(z_i)}{\sum_{i'=1}^{n} \exp(z_{i'})} \tag{4}$$

$$\alpha^i{}_{\mathrm{QT/COL}} = softmax\left(\mathbf{H^i}_{\mathrm{COL}}\mathbf{W^i}\mathbf{H^i}^{\top}_{\mathrm{QT}}\right) \tag{5}$$

$$\mathbf{H^i}_{\mathrm{QT/COL}} = \alpha^i{}_{\mathrm{QT/COL}}\mathbf{H^i}_{\mathrm{QT}} \tag{6}$$

This column attention mechanism is employed for all models, which notably enhanced performance compared to earlier models without column attention as reported in (Xu et al., 2017; Yu et al., 2018). As such, each question is conditioned on the column names. The column names here are similar to a question in a Question Answering (QA) task, where the sentences in a paragraph are conditioned on the question.

3.4 Ensembling

Recent studies have shown (Huang et al., 2017; Caruana et al., 2004) and the GLUE leaderboard (Wang et al., 2018a) indicates that ensemble models consistently outperform single models on a variety of optimization tasks. That is due to the fact that ensembles reduce the overall variance through the combination of models that converged to different local minima. Hence, overfitting can easier be avoided. Ensemble models are particularly superior to single models, if the individual members of the ensemble are accurate and diverse (Caruana et al., 2004). The latter can be achieved through training with different initialisations or learning rates.

To investigate the landscape of different ensemble versions, we trained both homogeneous and mixed ensembles. Homogeneous ensembles consisted of members that were initialised with the same word embeddings (i.e., either BERT or GloVe models), whereas members of mixed ensembles were trained with different word embeddings (i.e., BERT and GloVe models). Since training deep ensemble networks is computationally expensive (Huang et al., 2017) and time complexity increases linearly with the number of members, each of our ensemble models consisted of two members only (see Equation 7). Thus, we acknowledge the fact that this might not lead to a significant decrease in error rates. We suppose, however, that even two members yield lower variance, and converge to better local minima than a single model.

To yield an ensemble, the predictions of the individual members are combined either through majority voting (this is more suitable for classification tasks) or averaging. The latter can be computed as a simple arithmetic or weighted averaged (Krogh and Vedelsby, 1995). We combined individual models to an ensemble network through averaging with weights, which can be defined as follows.

$$\bar{M}(x) = \frac{1}{N}\sum_{y_i}^{N} w_{y_i} M^{y_i}(x) \tag{7}$$

$\bar{M}$ denotes the weighted ensemble average, where $N = 2$ is the number of ensemble members, y_i refers to the prediction or output of an individual model, x indicates the network's input, and w is our belief into an ensemble member. The more accurate the performance of a single model was, the higher was the value w with which we weighed the output of a model.

4 Results

4.1 Implementation

Our model was implemented in PyTorch (Paszke et al., 2017), exploited either context-free GloVe (Pennington et al., 2014) or contextual BERT embeddings (Devlin et al., 2018). The number of hidden units was set to 120 for each hidden layer. The same number of hidden layers, dropout rates, optimizer, and loss function as in (Yu et al., 2018) were used.[9]

4.2 Experiments

We performed numerous ablation studies to scrutinize which features add additional benefits, and which should rather be dropped prior to final testing. In initial experiments, we concatenated Part-of-Speech (POS) tag embeddings with Type embeddings and GloVe or BERT embeddings to represent a question. Note that this was done only for Model_COL and Model_OPVAL but not for Model_AGG. However, fine-tuning on the development set revealed that POS tags did not enhance the performance of our model. In fact, prediction accuracy for the WHERE clause was lower for models with POS embeddings compared to models without as can be inferred from Table 2. We suppose, that this is most likely due to the fact that both Type embeddings carry syntactic information (Yu et al., 2018), and contextual embeddings such as BERT have enough semantic information to infer low-level linguistic features such as POS tags (Peters et al., 2017, 2018; Devlin et al., 2018). Hence, we did not pursue this approach further, and focussed on our other models instead.

Results showed that all single models that exploited BERT embeddings performed significantly worse than the original GloVe models. Whereas the difference in the generation of the SELECT column between the best BERT single model (max-pool) and GloVe is only $\sim 3\%$, the difference between the accuracies for the WHERE clause are, with $\sim 8\%$, more notable.

			Dev		
Embeddings	Dim	Model	Acc_{agg}	Acc_{sel}	Acc_{where}
BERT (Avg) + POS	600	Single	0.896	0.883	0.647
BERT (Max) + POS	600	Single	0.898	0.888	0.659
BERT (Avg)	600	Single	0.897	0.886	**0.702**
BERT (Max)	600	Single	0.899	0.891	**0.700**

Table 2: Dev set results on the WikiSQL dataset for the different SQL clauses for BERT + POS vs. BERT without POS.

Moreover, ensemble models performed better than single models (1% absolute improvement), even when the members exploited lower-dimensional embeddings (100d) compared to the single models (600d). This, however, only holds for the generation of the SELECT clause, but not for the generation of the WHERE clause.

Higher dimensional word embeddings showed more benefits for BERT than for GloVe. GloVe 300d models showed an improvement of 0.7% for the SELECT and $\sim 1\%$ for the WHERE clause compared to GloVe 50d. In contrast, BERT 600d models showed an improvement of 3-4% for the SELECT and $\sim 3\%$ for the WHERE clause compared to BERT 100d. This might be due to the fact that BERT embeddings were extracted through Principal Component Analysis (PCA), and GloVe are pre-trained embeddings optimized via neural networks.

Overall results can be inferred from Table 3, and detailed results are displayed in Table 4. In each table, the letter on the right-hand side of an ensemble model indicates whether the ensemble was homogeneous (ensemble members utilized same embeddings) or mixed (ensemble members utilized different embeddings). PCA full denotes that all final embedding dimensions were extracted through PCA, whereas PCA split indicates that PCA 100 was performed on the last 268 dimensions only, and concatenated with the first 500 dimensions of the BERT embeddings.

Figure 1 depicts learning curves for the best BERT and GloVe single models. Accuracy plots over time show that the gap between training and validation accuracy is larger for BERT than for GloVe models. This means that BERT models overfit more than GloVe models. Moreover, not only does validation accuracy increase faster for GloVe than for BERT it also peaks at a later point in time (Epoch 87 for GloVe vs. Epoch 45 for

[9] Our code is available at https://github.com/LukasMut/TypeSQL-plus-BERT

			Dev			Test		
Embeddings	Dim	Model	Acc_{lf}	Acc_{qm}	Acc_{ex}	Acc_{lf}	Acc_{qm}	Acc_{ex}
BERT (Avg)	100 (PCA full)	Single	_	**0.556**	0.624	_	**0.548**	0.620
BERT (Avg)	100 (PCA full)	Ensemble (H)	_	0.554	**0.630**	_	0.544	**0.627**
BERT (Avg)	100 (PCA full)	Ensemble (M)	_	0.642	0.628	_	0.632	0.614
BERT (Max)	100 (PCA full)	Single	_	0.550	0.621	_	0.546	0.621
GloVe + Para	50	Single	_	**0.663**	**0.731**	_	**0.651**	**0.719**
GloVe + Para	50	Ensemble (H)	_	0.645	0.690	_	0.635	0.683
BERT (Avg)	600 (PCA full)	Single	_	**0.589**	0.654	_	0.582	0.653
BERT (Max)	600 (PCA full)	Single	_	0.585	**0.656**	_	**0.588**	**0.657**
BERT (Avg)	600 (PCA full)	Ensemble (H)	_	0.574	0.645	_	0.560	0.646
BERT (Max)	600 (PCA full)	Ensemble (M)	_	0.636	0.650	_	0.631	0.651
BERT (Avg)	600 (PCA split)	Single	_	0.580	0.648	_	0.575	0.646
BERT (Max)	600 (PCA split)	Single	_	0.584	0.650	_	0.576	0.645
BERT (Max)	600 (PCA split)	Ensemble (H)	_	0.574	0.649	_	0.575	0.647
BERT (Max)	600 (PCA split)	Ensemble (M)	_	0.647	0.640	_	0.638	0.635
GloVe + Para	300	Single (ours)	_	0.672	0.735	_	0.666	0.734
GloVe + Para	300	Single (Yu et al., 2018)	_	**0.680**	**0.745**	_	**0.667**	**0.735**
GloVe + Para	300	Ensemble (H)	_	0.650	0.552	_	0.644	0.542

Table 3: Overall results on the WikiSQL dataset. ACC_{lf}, ACC_{qm}, ACC_{ex} denote the accuracies for exact string (logical form), canonical representation, and execute result matches between the generated SQL query and the ground truth. All results are content-insensitive, that is only question and table schema (but not the content of the databases) are used as inputs for our model.

			Dev			Test		
Embeddings	Dim	Model	Acc_{agg}	Acc_{sel}	Acc_{where}	Acc_{agg}	Acc_{sel}	Acc_{where}
BERT (Avg)	100 (PCA full)	Single	0.899	0.868	0.676	0.899	0.857	0.669
BERT (Avg)	100 (PCA full)	Ensemble (H)	0.901	0.886	0.662	**0.901**	0.875	0.656
BERT (Avg)	100 (PCA full)	Ensemble (M)	**0.902**	**0.937**	0.738	0.900	**0.928**	0.732
BERT (Max)	100 (PCA full)	Single	0.900	0.864	0.672	0.897	0.848	0.670
GloVe + Para	50	Single	0.898	0.922	**0.774**	0.898	0.913	**0.769**
GloVe + Para	50	Ensemble (H)	0.901	0.936	0.742	**0.901**	0.925	0.740
BERT (Avg)	600 (PCA full)	Single	0.897	0.886	0.702	0.895	0.877	0.702
BERT (Avg)	600 (PCA full)	Ensemble (H)	0.899	0.895	0.682	0.901	0.884	0.671
BERT (Max)	600 (PCA full)	Single	0.899	0.891	0.700	0.902	0.883	0.702
BERT (Max)	600 (PCA full)	Ensemble (M)	0.900	0.934	0.735	0.900	0.927	0.733
BERT (Avg)	600 (PCA split)	Single	0.897	0.888	0.699	0.897	0.874	0.694
BERT (Max)	600 (PCA split)	Single	0.897	0.887	0.696	0.900	0.875	0.694
BERT (Max)	600 (PCA split)	Ensemble (H)	0.901	0.905	0.677	0.901	0.897	0.678
BERT (Max)	600 (PCA split)	Ensemble (M)	0.902	0.934	0.748	0.902	0.925	0.743
GloVe + Para	300	Single (ours)	0.899	0.929	0.783	0.900	0.919	**0.780**
GloVe + Para	300	Single (Yu et al., 2018)	**0.903**	0.931	**0.785**	**0.905**	0.922	0.778
GloVe + Para	300	Ensemble (H)	0.900	**0.939**	0.751	0.901	**0.930**	0.748

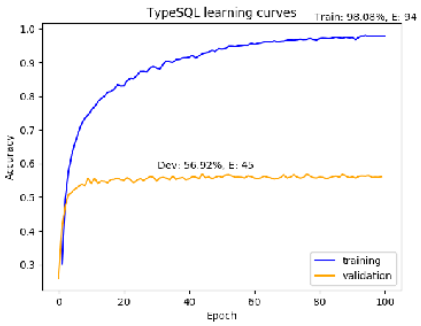
Table 4: Detailed results on the WikiSQL dataset for the different SQL clauses. ACC_{agg}, ACC_{sel}, ACC_{where} denote the accuracies for canonical representation matches on AGGREGATOR, SELECT, and WHERE clause respectively.

BERT). Loss plots over time, however, appear to be highly similar for both models and show that training worked reasonably well for BERT as for GloVe.

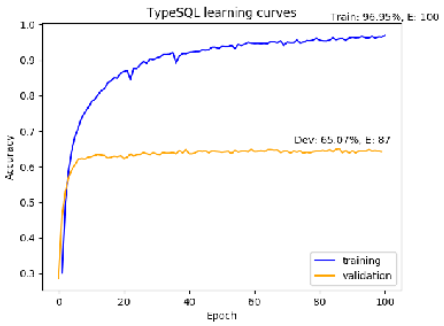
5 Discussion

We suppose that BERT models, although exploiting contextual embeddings, performed worse than context-free GloVe models due to the following reasons.

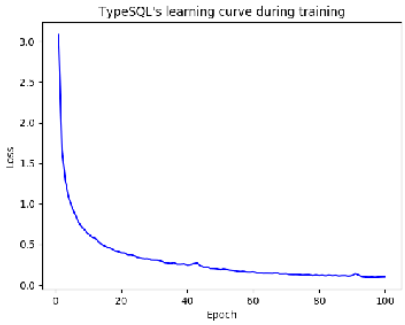
Firstly, we did not utilize full-dimensional (768) BERT embeddings. Owing to the fact that we resorted to GloVe embeddings for Model_AGG, we were forced to use the same dimensions for BERT as for GloVe plus paraphrase embeddings. Although PCA captures features that explain the most variance in the data, a vast part of contextual information might get lost when performing dimensionality reduction on BERT. BERT vectors are highly optimized contextual



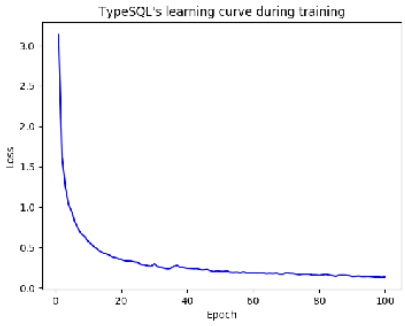
(a) Overall accuracy for 600d BERT



(b) Overall accuracy for 600d GloVe



(c) Loss for 600d BERT



(d) Loss for 600d GloVe

Figure 1: Learning curves over 100 epochs for 600d BERT (*max-pool*) and 600d GloVe single models

word embeddings for which the most important word features have been extracted before through transformer networks. Thus, additional dimensionalty reduction will remove decisive information. This can be inferred from the fact that higher-dimensional BERT models performed notably better than lower-dimensional BERT models, whereas the difference for GloVe models was not as large (see Table 3). Hence, we encourage future studies in the field of NL2SQL to use full-dimensional BERT embeddings to exploit the entire scope of contextual information.

Secondly, contextual information might have gotten lost already prior to dimensionality reduction through retokenization. We either averaged or max-pooled BERT embeddings on sub-word level to rejoin WordPiece tokens into full word tokens (see Algorithm 1). Hence, we could not retain thorough information on sub-word or character level. To overcome the latter problem, we will, in future work, modify the network architecture to not be forced to rejoin WordPiece tokens into full word tokens.

Interestingly, model ensembles compared to single models performed better for the generation of the SELECT clause but worse for predicting the WHERE clause. The higher prediction accuracy for the SELECT clause is reasonable given the fact that model ensembling reduces variance and thus overfitting, and as such makes models more robust (Huang et al., 2017; Izmailov et al., 2018). However, we have no thorough explanation for worse performances on the WHERE clause. Predicting the WHERE clause is the most difficult among the three and highly complex (Xu et al., 2017; Yu et al., 2018). It requires the generation of multiple values (i.e., number of columns, operation value, condition value - the latter two as often as there are additional AND clauses) compared to just a single value for AGGREGATE and SELECT respectively. Perhaps, predictions from different models differ too significantly from each other, and hence averaging over model predictions leads to worse query generations than the WHERE prediction from a single model. This might be interesting to inspect in future NL2SQL experiments.

6 Future work

Due to time constraints, and limited resources we could, unfortunately, not investigate all architectures we would have liked to examine. One such

intriguing architecture is the recently developed SQLova model (Hwang et al., 2019). Owing to the promising results reported in their paper, we believe it is reasonable to scrutinize their approach further in future work. What we deem particularly interesting is their way of separating SQL table headers during the generation of the SQL query. Similarly to our approach, SQLova utilizes BERT, and hence denotes the beginning of a question by the special [CLS] token and separates a question from a query by the special [SEP] token. Moreover, SQLova separates each table header from another by a [SEP] token to exploit the fact that the order of the column names does not matter, and each column name in the query corresponds to a single SQL keyword (i.e., SELECT, WHERE or AND). This appears to additionally help the decoder in the query generation task. Thus, we aim to incorporate this mechanism in future experiments.

7 Conclusions

Three conclusions in particular can be drawn from our work. Firstly, BERT does not help TypeSQL to generate a query in the way we have implemented it. Even though our results did not provide evidence to use BERT rather than GloVe, we still believe that contextual embeddings can enhance the performance of NL2SQL algorithms. Most probably, one must not rejoin the byte-pair encoded sub-words into full word tokens such as we performed, but rather make use of every single WordPiece token in order to exploit the entire range of contextual features, and not lose any information.

Secondly, embeddings with lower dimensionality did not lead to notably worse results than higher dimensional word vectors. This holds for both BERT and GloVe, although more for GloVe as lower-dimensional GloVe compared to BERT were particularly optimized through neural networks. From this observation, one could infer that future work in the realm of pre-training unsupervised language models for downstream NLP tasks should rather shed light toward the essential features of a contextual word representation instead of computing ever higher-dimensional vectors. The latter could help to save memory, and reduce computational time.

Lastly, ensemble models yield more accurate generations than single models. This, however, holds only for the prediction of the SELECT clause.

Acknowledgements

I would like to thank my supervisor Johannes Bjerva for the weekly meetings and the fruitful comments throughout.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM.

Xingyi Cheng, Weidi Xu, Kunlong Chen, Wei Wang, Bin Bi, Ming Yan, Chen Wu, Luo Si, Wei Chu, and Taifeng Wang. 2019. Symmetric regularization based bert for pair-wise semantic reasoning. *arXiv preprint arXiv:1909.03405*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*.

Michael Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, GP Bhargav, Dinesh Garg, and Avirup Sil. 2019. Span selection pre-training for question answering. *arXiv preprint arXiv:1909.04120*.

Tong Guo and Huilin Gao. 2017. Bidirectional attention for sql generation. *arXiv preprint arXiv:1801.00076*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. 2017. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Anders Krogh and Jesper Vedelsby. 1995. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

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. In *NIPS Autodiff Workshop*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

Adwait Ratnaparkhi. 1997. A simple introduction to maximum entropy models for natural language processing. *IRCS Technical Reports Series*, page 81.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018a. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2018b. Pointing out sql queries from text.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

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

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Zhuosheng Zhang, Yuwei Wu, Hai Zhao, Zuchao Li, Shuailiang Zhang, Xi Zhou, and Xiang Zhou. 2019. Semantics-aware bert for language understanding. *arXiv preprint arXiv:1909.02209*.

Kai Zhao and Liang Huang. 2014. Type-driven incremental semantic parsing with polymorphism. *arXiv preprint arXiv:1411.5379*.

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