Building Smaller & Faster Text-to-SQL Models

Le Hoang Van
Georgia Institute of Technology
vhoang31@gatech.edu

Abstract

Research in the domain of text-to-SQL translation has recently gathered steam, with the advent of neural architectures such as Microsoft's RAT-SQL. Taking the reported RAT-SQL+BERT architecture as our baseline and Spider as our dataset, we performed several ablative experiments, with the aim of achieving the performance reported in the original paper, while reducing the overall complexity of the architecture. Our approach focused on experimenting with combinations of simpler embedding models, DistilBERT and TinyBERT, with the original RAT-SQL architecture and its ablated variant - DuoRAT. Our experiments conclude that our simpler architectures required significantly lesser training iterations over the dataset to converge close to the original paper's result. Our experiment results also point to the efficacy of DistilBERT and TinyBERT in nearly matching the performance of BERT on external tasks despite significant reduction in complexity.

1. Introduction

Translating natural language questions to SQL is both a novel and challenging task. Research in this area has recently commanded the attention of several research teams, who have attempted to tackle this challenging task by training complex neural architectures on the Spider dataset. One such architecture is Microsoft's RAT-SQL introduced in July 2020. While it is no longer the state-of-the-art, it is a popular, highly regarded and reproducible baseline. We reproduced the results of the RAT+BERT architecture reported in the original paper and then performed several experiments of an ablative nature, with the goal of achieving the original architecture's performance with significantly simpler and faster models.

Generally, the approach to produce a state-of-the-art result on a dataset is to increase the complexity of a baseline architecture. The architectures on the leaderboard of the Spider dataset attest to this trend. This trend poses several challenges, one of which is that the trend of increasing complexity generally tends against the democratization and reproducibility of research. In some cases, only research teams with highly specialized and expensive compute resources are able to produce, utilize and lay claims to state-of-the-art performance. Hence, we hope that our work will not only demonstrate the ability of simpler models to achieve benchmark results on the aforementioned specific cross-domain translation task, but it will also elevate the overall importance of such research aimed at reducing complexity and inspire other research teams to dedicate time and resources in the direction of increasing the democratization and reproducibility of research.

We used the Spider [19] dataset for our experiments. It is a complex text-to-SQL dataset that comprises of 10,181 natural language questions and 5,693 corresponding SQL queries on 200 databases. Unlike WikiSQL [20] which lacks multi-table schemas, the examples in Spider involve complex nested queries, set operations, and multiple tables joining via foreign keys. This last property of the Spider dataset presents the challenge of schema linking which RAT and, by extension, DuoRAT, are designed to solve. The SQL queries in Spider are divided into 4 levels of difficulty: *Easy*, *Medium*, *Hard*, and *Extra Hard*, defined based on the number of SQL components, selections, and conditions, so that queries that contain more SQL keywords, nested subqueries, column selections and aggregators, etc. are considered harder. Table 1 gives an example of an *Extra Hard* query in the Spider dataset.

As the authors have made the test set available only through an evaluation server, we perform all evaluations using the latest development set that was released on June 7, 2020. It contains 1,034 examples, with databases and schemas distinct from those in the training set.

2. Approach

2.1. Relation-aware self attention

A self-attention encoder, or Transformer [15] is a stack of self-attention layers where each layer consists of H attention heads. Each attention ahead operates on an input sequence $x = (x_1, \ldots, x_n)$ where $x_i \in \mathbb{R}^{d_x}$, and computes a new sequence $z = (z_1, \ldots, z_n)$ where $z_i \in \mathbb{R}^{d_z}$ as fol-

Question	Find the first and last names of the students who are living in the dorms that have a TV Lounge as an amenity
Database	database with 5 tables e.g. `student`, `dorm_amenity`, etc.
Annotation	`SELECT t1.fname, t1.lname FROM student AS t1 JOIN lives_in AS t2 ON t1.stuid WHERE t2.dormid IN (SELECT t3.dormid FROM has_amenity AS t3 JOIN dorm_amenity AS t4 ON t3.amenid = t4.amenid WHERE t4.amenity_name='TV Lounge')`

Table 1. An example of an *Extra Hard* utterance-database-SQL triplet in the Spider dataset.

lows:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^\top}{\sqrt{d_z}}, \quad \alpha_{ij} = \underset{j}{\text{softmax}}\{e_{ij}\}, \\ z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V), \tag{1}$$

where e_{ij} is the attention weight from position j to i, $\sqrt{d_z}$ is a scaling factor, and $W_Q, W_K, W_V \in \mathbb{R}^{d_x \times d_z}$ are weight matrices corresponding to queries, keys, and values, respectively. In the original self-attention formulation, sinusoidal functions are used with transformers to parameterized position embeddings in a fixed ad hoc way and are added to encoder and decoder inputs prior to the first layer. More recently, [13] proposed an extension to self-attention to consider the pairwise relationship between input tokens, called relation-aware self-attention, by adding the fully learnable directed labeled edge vectors r_{ij}^K, r_{ij}^V to Equations 1 as follows:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K + r_{ij}^K)^\top}{\sqrt{d_z}}, z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + r_{ij}^V) \tag{2}$$

2.2. RAT-SQL

Relation-aware Transformers, or RAT [16] , is the first unified framework for embedding hard relational structure and "soft" induced relations jointly when learning contextual representations with Transformers. While Transformers excel at learning "soft" relations in unstructured data, graph neural networks [2] excel at learning patterns over existing "hard" relational structures, and the RAT framework combines the representational power of both. An instantiation of this framework with application in text-to-SQL parsing is aptly called RAT-SQL. At a high level, the system consists of as an encoder which consumes the input graph as a joint modeling of question and schema, and a decoder which produces the output abstract syntax tree (AST).

2.2.1 Question-Schema joint encoding

We view each question/schema token as a node in a graph. The most basic edge type among the nodes is a generic link between any pair of tokens, reflecting the assumption that a priori any token could provide relevant context to any other token, so a link cannot be ruled out. This essentially yields a fully connected graph.

However, other types of relations carry special meanings and are sparse. These include (i) scheme-linking relations, which provide explicit alignment between words in the question and words in column or table names, (ii) table-column relations, which encode information about the layout of the schema, and (iii) foreign-key relations, which indicate the link between a column in one table to the primary key of another table. These are added to the graph using relational edge vectors as shown in Equations 2. Because there can be more than one type of edge between two tokens to be modeled, we now have a multi-graph input.

To formalize what the encoder does, we define by $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ the schema elements, consisting of tables and columns, and $Q = q_1 \ldots q_{|Q|}\}$ the sequence of words in the question. Let $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ be the multi-graph with edge set $\mathcal{E}$ and node set $\mathcal{V}$. The encoder maps $\mathcal{G}$ to a joint representation $\mathcal{H} = \{\phi_1^q, \ldots, \phi_{|Q|}^q\} \cup \{\phi_1^s, \ldots, \phi_{|\mathcal{S}|}^s\}$ in a two-stage process. In the first stage, the fully-connected portion of the graph is flattened into a linear string, with special tokens added to identify whether the token belongs to a question or a column/table, before being fed to a pre-trained model such as BERT. The use of pre-trained language model hers is how implicit common sense knowledge is embodied in the semantic parser. Then in the second stage, we feed the BERT output embeddings into a RAT, given by Equation 2, to model information propagation along the special sparse edges of the multi-graph.

2.2.2 Grammar-based decoding

If we model SQL queries as linear sequences of text tokens on the output side à la machine translation, it is not easy to leverage on SQL grammar knowledge. In contrast to machine translation task where giant text corpora are readily available for training, text-to-sql data is scarce and usually requires human annotation effort. Thus, leveraging on prior knowledge of SQL grammar will significantly improve the learning efficiency even with few examples. Therefore, we want to cast the problem as generating the AST of SQL queries.

A common approach to predict an AST is to use a grammar-based transition system like TranX ([18]), which

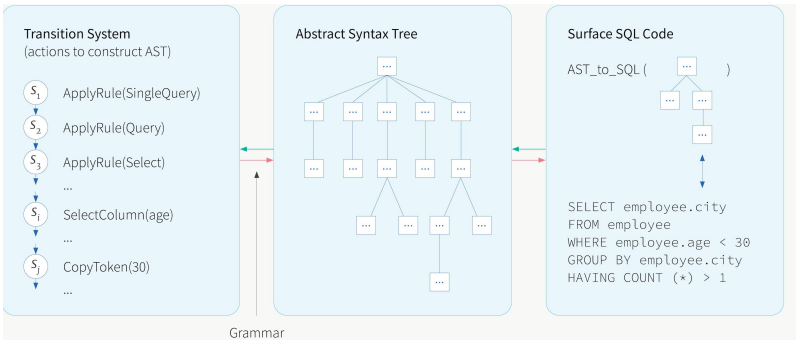


Figure 1. How the TranX framework makes symbolic grammar knowledge accessible to a neural decoder: instead of directly predicting tokens of surface SQL code, the model would predict the sequence of AST-constructing action sequences. Before training, we preprocess the SQL code string by transforming it into its AST ($B \leftarrow C$). Then the TranX transition system builds the action sequences ($A \leftarrow B$) guided by the SQL grammar, expressed in the abstract syntax description language of TranX (G). Training can be done via maximum likelihood sequence learning with parent-feeding. During inference, the model predicts action sequences guided by the grammar and the transition system, which defines the set of legal generation at each step. The generated action sequences are then post-processed deterministically to form the surface SQL code strings ($A \rightarrow B \rightarrow C$). Source: [3]

decomposes the generation process of an AST into a sequence of actions. The neural network learns to predict the action sequence, and the transition system then constructs the AST using the predicted action sequence. Finally, another deterministic routine maps the AST into a linear string format of SQL, also called the surface code.

The SQL grammar is specified in an abstract syntax description language ([17]) and is specific to the programming language of interest, and needs to be developed by a human expert. The transition system converts between an AST and its AST-constructing action sequence, leveraging the grammar above. The transition system starts at the root of the AST and derives the action sequence by a top-down, left-to-right, depth-first traversal of the tree. At each step, it generates one of the possible action types (Figure 1). The decoder is implemented as an LSTM network with parent feeding to reflect the topological nature of an AST. The LSTM decoder retrieves a context vector from the input encodings using multi-head attention.

2.3. DuoRAT

Unlike RAT-SQL which only uses a transformer in the encoder, both the encoder and decoder of DuoRAT [12] are transformers. Encoders of RAT and DuoRAT are almost identical except the schema-linking heuristic they use. Although both distinguish between name-based schema linking (NBSL) and content-based schema linking (CBSL), RAT's allows three modes of matching, namely *ExactMatch*, *PartialMatch*, and *NoMatch*, while DuoRAT's simplifies everything to a partial match with confidence c where c is binary and either *high* or *low*.

To mimic the use of parent-feeding in RAT's LSTM decoder, the TranX framework for grammar-constrained sequence prediction [18] has been extended to work with relation-aware transformers. Like in the original framework, the decoder is restricted to generate only those sequences of grammar actions that encode a valid SQL AST. RAT-SQL and the TranX framework use a custom parent-feeding LSTM decoder where the LSTM is fed also its own state from a previous step at which the constructor of the current action's parent AST node was generated. By contrast, in DuoRAT's relation-aware transformer decoder, the relations are derived from the structure of the SQL program code. There are four types of self-attention decoder relations: *(i) parent-child* and *(ii) child-parent*, to encode whether node is parent or child of a node in the decoded AST, respectively; *(iii) identity* to connect a node in the decoded AST to itself; and *(iv) sibling-distance* d, to inform relative distance of nodes having the same parent. The decoder is coupled to the encoder via a relation-aware memory attention mechanism. Here, relations are used to indicate which tokens from the input were copied to the output, that is, either question tokens, tables, or columns, depending on the type of the literal that was produced.

Unlike RAT's LSTM decoder which uses scaled dot product attention to compute the context representation, DuoRAT's transformer decoder uses Bahdanau attention [1] where the alignment function is modeled as a neural network. A more efficient implementation of Bahdanau attention is introduced to prevent out-of-memory issue when the encoder/decoder sequence is too long.

2.4. DistilBERT and TinyBERT

Knowledge distillation aims to transfer the knowledge embedded in a large teacher network to a small student network where the student network is trained to reproduce the behaviors of the teacher network. Using the teacher signal, [11] trained a smaller language model, called DistilBERT,

System	#Params	Speedup	Avg
$\text{BERT}_{\text{BASE}}$ (teacher)	109M	1.0x	79.5
DistilBERT_6	67.0M	2.0x	76.8
TinyBERT_6	67.0M	2.0x	79.4

Table 2. Results are evaluated on the test set of GLUE official benchmark; only the averaged score is shown for brevity, please refer to [6] for task-specific scores. Both student networks have number of layers $N = 6$, hidden size $d = 768$, feedforward/filter size $d_i = 3072$, and number of attention heads $h = 12$. The original $\text{BERT}_{\text{BASE}}$ ($N = 12, d = 768, d_i = 3072, h = 12$) is used as the teacher model.

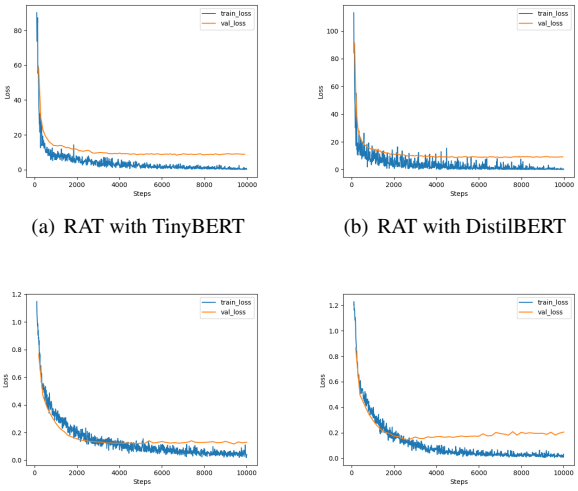
from the supervision of BERT. Following [5], the training loss is a linear combination of the distillation, masked language modeling, and cosine-distance losses. The student is a small version of BERT in which the token type embeddings and the pooler (used for the next sentence classification task) are removed and the number of layers reduced by a factor of two.

More recently, [6] proposed a novel Transformer distillation method that is specially designed for knowledge distillation of Transformer-based models. By running knowledge distillation twice, once with the original pre-trained BERT model as teacher, and then again with task specific fine-tuned BERT as a new teacher, this framework ensures that the student, TinyBERT, captures both general domain as well as task specific knowledge in BERT. Table 2 compares two variants of DistilBERT and TinyBERT with similar architecture on the GLUE official benchmark.

3. Reducing training time

For brevity, for the rest of this report, we will be referring to RAT-SQL as simply RAT. [12] reported a training time of six and two days for RAT and DuoRAT respectively, using $\text{BERT}_{\text{LARGE}}$ pre-trained model on a single V100 with 32 GB HBM2. Since most of us could only procure 16 GB GPUs due to Google Cloud Platform (GCP)'s constraints, it would be impossible to run a sufficient number of experiments and produce a report within the stipulated deadline. Thus we explored clipping training steps to reduce training time. The first experiment we performed is to replicate the RAT + $\text{BERT}_{\text{LARGE}}$ experiment as described in [16], to establish a baseline and also get a sense of the training time for our specific GPU setup. In order to obtain same accuracy scores in the paper, several important modifications needed to be made, please refer to Appendix A for details. We trained the model for up to 80,000 steps and evaluated every 10,000 steps using Spider's development set. We noticed that most of the learning happens in the first 10,000 steps since there's only a modest gain in accuracy after that (55.2% at 10,000 steps versus 70.1% at 80,000 steps). We hypothesized DuoRAT + $\text{BERT}_{\text{LARGE}}$ shows the same behavior so we ran it with default configurations for the first 10,000 steps. We compared the accuracy of this model checkpoint (64.2%) with the original model (69.9%) and agreed that for subsequent experiments, 10,000 training steps is sufficient.

4. Experiments and Results



(a) RAT with TinyBERT (b) RAT with DistilBERT

(c) DuoRAT with TinyBERT (d) DuoRAT with DistilBERT

Figure 2. Training and Validation Loss Curves for Different Model Configurations

4.1. Loss function and training scheme

All the experiments involving RAT and DuoRAT will be using learning rate schedulers with implementations shown in the Appendix B's Algorithms 1 and 2 respectively. Within the encoder, we use WordPiece tokenization. All word embeddings have dimension 300. As for loss functions, RAT and DuoRAT would be using softmax and negative log likelihood respectively. Both loss functions are conceptually the same but they expect different inputs and generates outputs on different scales which could be seen in Figure 2. Also, DuoRAT's experiments will be using Adam [7] as its optimizer throughout. Elaborating further, the choice argument in Appendix B's Algorithms 1 and 2 would state the model whose learning rate is being updated at the current step.

In our experiments, we have observed the occurrence of an interesting phenomenon named Deep Double Descent which refers to the situation where test error first decreases then increases near the interpolation threshold and decreases thereafter [9]. Further elaboration on this can be seen in the Appendix C. A plot summarising the results of the best configurations for all models can also be found in Appendix E.

Model	*RAT + BERT	RAT + BERT	RAT + DistilBERT	RAT + TinyBERT	*DuoRAT + BERT	DuoRAT + BERT	DuoRAT + DistilBERT	DuoRAT + TinyBERT
Split								
Easy	86.4	77.4	**89.5**	82.7	-	87.5	83.1	79.4
Medium	**73.6**	55.2	69.5	69.1	-	64.3	58.3	53.4
Hard	**62.1**	43.1	58.3	56.3	-	55.2	50.6	46.6
Extra Hard	42.9	34.9	44.1	**49.4**	-	38.6	33.1	36.7
All	69.7	55.2	63.9	67.0	**69.9**	64.2	58.9	55.8

Table 3. Accuracy scores of various models on the Spider development set, by difficulty. The results marked with * correspond to the results obtained by training RAT + BERT and DuoRAT + BERT till 80000 and 100000 steps respectively.

Model	RAT + BERT	RAT + DistilBERT	RAT + TinyBERT	DuoRAT + BERT	DuoRAT + DistilBERT	DuoRAT + TinyBERT
Metric						
Accuracy	55.2	63.9	67.0	64.2	58.9	55.8
Performance Loss (%)	0	-15.8	-21.4	-16.3	-6.7	-1.1
Speedup	1.0x	1.5x	1.4x	2.3x	6.8x	8.2x

Table 4. Accuracy and speedup comparisons with RAT + BERT as the baseline

We use DistilBERT_6 and TinyBERT_6 as described in Table 2 in all of our subsequent experiments. For brevity, $\text{BERT}_{\text{LARGE}}$, DistilBERT_6, TinyBERT_6, will hereinafter be referred to as BERT, DistilBERT, and TinyBERT, respectively, unless otherwise noted.

4.2. Evaluation metric

We report exact match accuracy as the main evaluation metric. [19] measures the exact matching by evaluating whether the predicted query as a whole is syntactically equivalent to the "gold" (or target) query without considering the returned value after execution. The predicted query is correct only if all of its components matches those from the gold query. Each query is decomposed into components such as `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY`, `KEYWORDS` (including all SQL keywords except column and table names and operators), and each component is subsequently decomposed into bags of sub-components. For example, consider a `SELECT` component such as `SELECT AVG(col1), MAX(col2), MIN(col1)` which can be parsed and decomposed into a set { `(AVG, MIN, col1), (MAX, col2)` }. To handle the ordering issue, we use set operations to check whether the predicted and gold sets are the same.

We report results obtained with beam search where beam size is 1.

4.3. Results

All 4 experiments were run on separate Google Cloud Platform servers each with a single NVIDIA T4 16GB GPU.

As discussed above, DuoRAT and DistilBERT/TinyBERT are touted as being smaller and faster to train than their respective predecessors, RAT and BERT, while achieving comparable performance. We therefore hypothesize at the outset that we would see such a trend in all our experiments. The only difference we expect would be the magnitude of speedup and performance degradation with respect to the baseline (i.e. RAT with BERT). Table 3 and 4 summarize our findings and Figure 2 shows the training/validation loss curves for the best training run in each of our experiments, in which no overfitting was detected.

Of all the experiments, RAT with TinyBERT achieved the highest exact match accuracy on the validation set while displaying the lowest speedup of 1.4x. Each training run took about 14.5 hours and only involve varying the batch size and TinyBERT's fine-tune learning rate while keeping all other hyperparameters constant. We found that batch sizes smaller than 12 severely hinder the model's ability to learn, and is in fact more important than learning rate, at least for the first 10,000 training steps. We experimented with various learning rates ranging between 2e-06 to 6e-05, and even with such a wide range, the model consistently achieves over 64% in accuracy, eventually peaking at 67% when the learning rate is at the top of the range. However, increasing the learning rate any further than this (1e-04) and the accuracy quickly plummets to 60%, and the learning curve also shows large volatility.

RAT with DistilBERT came in 2nd with the best validation accuracy at 63.9%, slightly lower than RAT with TinyBERT. The amount of speedup is also similar to the latter at 1.5x of the benchmark. Of all hyperparameters that

could be tuned, batch size and initial learning rate yielded the most variation in results. A batch size of 2 produced a better performance of 63.9%, compared to larger batch sizes of 6 and 8, which yielded 63.2% and 63.3% respectively. Smaller batch sizes seemed to introduce noise into the training, which might have resulted in the model with a small batch size of 2 reducing the overfitting and generalizing better. For all experiments, the learning rate scheduler of the original implementation was preserved. The default initial learning rate of 7.4e-04 as reported in the original paper works best. A faster initial learning rate of 7.4e-02 does not converge as the model weights are experiencing large amplitudes of variation about the minima in their magnitude. Comparatively, a slower rate of 7.4e-06 takes too long to converge.

DuoRAT with DistilBERT came in third with a validation accuracy of 58.9%, significantly less than RAT with DistilBERT. However, the former achieved a rather drastic speedup of 6.8x over the benchmark with each training run taking about 3 hours for 10,000 steps. Interestingly, the exact hyperparameter configuration used in the original paper [12] yielded the best accuracy. We experimented with various hyperparameter configurations but none gave a better accuracy than the default configuration. In fact, even slight increases in learning rates, e.g. increasing from 0.0001 to 0.0003 led to a drastic drop in exact-match accuracy from 58.9% to 21.9%. Given that overfitting had not occurred, we expected a slight improvement in accuracy, not a drastic reduction. Changes in other hyperparameters did not negatively impact the accuracy to the same extent though they did lead to 2%-5% fall in accuracy.

Coming in fourth in terms of validation accuracy is DuoRAT with TinyBERT with 55.8%. Despite having the lowest accuracy, this combination produced the highest speedup of 8.2x over the benchmark RAT with BERT with each training run taking about 2.5 hours. The run yielding the best accuracy was performed with the default configuration except that batch size of 27 was used and initial learning rate is set to be 0.0005 with a learning rate schedule as defined in Algorithm 1 found in Appendix B.

From our results above, what stood out were the relatively low accuracies achieved by DuoRAT with DistilBERT and TinyBERT (58.9% and 55.8% respectively). Both did not exceed 60% and the gap between the 1st-placed RAT with TinyBERT and 4th-placed DuoRAT with TinyBERT was a rather-large 11.2 percentage points. This certainly merits further investigation in future research.

4.4. Error Analysis

Since RAT with TinyBERT provides the greatest balance between performance and training time, we conducted our error analysis based on the evaluation results of this model combination. This would allow us to understand when the model fails and help to inform further enhancements.

We identified three main issues jeopardising the performance of these models as follows: (1) Ineffective matching and detection of column names in utterances: Missing column selector is a common occurrence during our analysis, even in *Easy* queries where there are explicit mentions of the columns in the utterance. (2) Failure in inferring columns from cell values through implicit means: It is difficult for the models to deduce the column name when encountering corresponding cell values in the utterance. (3) Inability to compose complex queries: Despite notable effort in alignment between question and schema tokens, it is a non-trivial task for models to construct complex target SQL queries and matching their selected columns to the correct clauses. An example of a complex query would be when it is a nested one. Examples related to these 3 issues can also be seen in Appendix D.

5. Conclusion

In this paper, we experimented with combinations of simpler embedding models - DistilBERT and TinyBERT- with the original RAT-SQL architecture and its ablated variant - DuoRAT. We showed that more lightweight language models distilled from BERT can be successfully fine-tuned with RAT architecture to produce text-to-SQL parsers that under some low resource settings can approximate the accuracy of, and in RAT-SQL's case, even outperform those trained with the original BERT models.

In our future work, we will combine both developments in knowledge distillation and further simplifications of the RAT-SQL framework to continue building low-resource yet high-performing text-to-SQL models. Not long after TinyBERT was released, we already have a successor that claims to provide even greater speedup without significant performance loss [4]. Another promising direction is to use a pretrained encoder-decoder pair as proposed in [10] and [8]. Base on our error analysis, we would want to look into frameworks that better capture the alignment between utterrances and scheme/table contents, such as [14].

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation. 3

[2] Ben Bogin, Matt Gardner, and Jonathan Berant. Representing schema structure with graph neural networks for text-to-sql parsing. *CoRR*, abs/1905.06241, 2019. 2

[3] Yanshuai Cao. Why is cross-domain text-to-sql hard. Accessed: 2021-08-01. 3

[4] Xuanang Chen, Ben He, Kai Hui, Le Sun, and Yingfei Sun. Simplified tinybert: Knowledge distillation for document retrieval. *CoRR*, abs/2009.07531, 2020. 6

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 4

[6] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351, 2019. 4

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 4

[8] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019. 6

[9] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *CoRR*, abs/1912.02292, 2019. 4, 8, 9

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. 6

[11] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. 3

[12] Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. Duorat: Towards simpler text-to-sql models. *CoRR*, abs/2010.11119, 2020. 3, 4, 6

[13] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018. 2

[14] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. *CoRR*, abs/2012.10309, 2020. 6

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 1

[16] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. *CoRR*, abs/1911.04942, 2019. 2, 4, 8

[17] Daniel C. Wang, Andrew W. Appel, Jeffrey L. Korn, and Christopher S. Serra. The zephyr abstract syntax description language. In Chris Ramming, editor, *Proceedings of the Conference on Domain-Specific Languages, DSL'97, Santa Barbara, California, USA, October 15-17, 1997*, pages 213–228. USENIX, 1997. 3

[18] Pengcheng Yin and Graham Neubig. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 7–12. Association for Computational Linguistics, 2018. 2, 3

[19] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019. 1, 5

[20] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. 1

A. RAT + BERT Experiment Setup

Similar to [16], we stack 8 relation-aware self-attention layers on top of the birectional LSTMs. Within them, we set $d_x = d_z = 256, H = 8$, and use dropout with rate 0.1. The position-wise feed-forward network has inner layer dimension 1024. Inside the decoder, we use rule embeddings of size 128, node type embeddings of size 64, and a hidden size of 512 inside the LSTM with dropout of 0.21.

Unlike the paper (and the accompanying code), we fine-tune BERT with a learning rate of 1e-05 (instead of 3e-06), and use softmax loss with label smoothing instead of just softmax loss, after consulting with the authors. The above changes helped us to train a model that reaches a similar accuracy as the one shown in the paper, which is also *RATSQL v3 + BERT (DB content used)* on the official Spider leaderboard at `https://yale-lily.github.io/spider`. The use of label smoothing is especially important in RAT-SQL experiments since it alleviated the problem of gradient explosion as reported by not only ourselves but also others who attempted to run the code with the default configuration.[1,2,3]

B. Learning Rate Schedulers of RAT and DuoRAT

Algorithm 1 RAT's Learning Rate Scheduler

procedure LR_SCHEDULER(current_step, num_warmup_steps, start_lr, end_lr, decay_steps, power, choice)
  if current_step $<$ num_warmup_steps **then**
    if choice == "BERT" **then**
      final_lr $= 0$
    else
      final_lr $=$ start_lr $*$ $\frac{\text{current\_step}}{\text{num\_warmup\_steps}}$
  else
    final_lr $=$ (start_lr $-$ end_lr) $*$ $\left(1 - \frac{\text{current\_step - num\_warmup\_steps}}{\text{decay\_steps}}\right)^{\text{power}}$ $+$ end_lr
  return final_lr
end procedure

C. Deep Double Descent

In the course of our experiments, we encountered 'deep double descent' phenomenon in around 70% of our experiments [9]. Figures 3 and 4 show two examples.

[9] introduced a generalized double descent hypothesis: models and training procedures exhibit atypical behavior when their Effective Model Complexity (ECM) is compara-

[1]https://github.com/microsoft/rat-sql/issues/3
[2]https://github.com/microsoft/rat-sql/issues/7
[3]https://github.com/microsoft/rat-sql/issues/11

Algorithm 2 DuoRAT's Learning Rate Scheduler

procedure LR_SCHEDULER(current_step, num_warmup_steps, start_lr, end_lr, decay_steps, power, factor, choice)
  if current_step $<$ num_warmup_steps **then**
    lr $=$ start_lr $*$ $\frac{\text{current\_step}}{\text{num\_warmup\_steps}}$
  else if current_step $<$ (num_warmup_steps $+$ decay_steps) **then**
    lr $=$ (start_lr $-$ end_lr) $*$ $\left(1 - \frac{\text{current\_step - num\_warmup\_steps}}{\text{decay\_steps}}\right)^{\text{power}}$ $+$ end_lr
  else
    lr $=$ end_lr
  if choice == "BERT" **then**
    final_lr $=$ lr / factor
  else
    final_lr $=$ lr
  return final_lr
end procedure

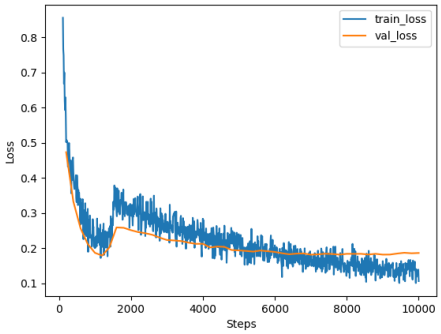


Figure 3. DuoRAT with DistilBERT, learning rate of 0.0005 and other default hyper-parameters

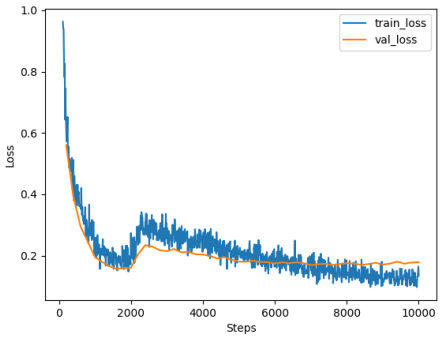


Figure 4. DuoRAT with DistilBERT, learning rate of 0.0003 and other default hyper-parameters

ble to the number of train samples [9]. In our experiments, our dataset is relatively small while DuoRAT has relatively high complexity - these are likely contributing factors toward "model-wise double descent" described by [9].

D. Error Examples From Spider Development Set

Issue 1	Failure in matching and detection of column names in utterances.
Utterance	Show the name and the release year of the song by the youngest singer.
Prediction	`SELECT singer.Name, singer.Song_release_year FROM singer ORDER BY singer.Age Desc LIMIT 1`
Error	Missing column `song_name` in `SELECT` clause and erroneous column selection.
Issue 2	Misinference of columns based on cell values.
Utterance	What are the dog name, age and weight of the dogs that were abandoned? Note that 1 stands for yes, and 0 stands for no in the tables.
Prediction	`SELECT Dogs.name, Dogs.age, Dogs.weight FROM Dogs WHERE Dogs.breed_code = 'terminal' AND Dogs.date_departed = 'terminal'`
Error	Missing `abandoned_yn = 1` and erroneous `WHERE` clause
Issue 3	Failure in composing complex target SQL.
Utterance	Find the name of airports which do not have any flight in and out.
Prediction	`SELECT airports.AirportName FROM airports WHERE airports.AirportCode NOT IN (SELECT flights.SourceAirport FROM flights)`
Error	Should include `UNION SELECT DestAirport FROM Flights` in the subquery.

Table 5. Error examples collected from the SPIDER development set based on RAT + TinyBERT

E. Summary of Best Configurations for All Models

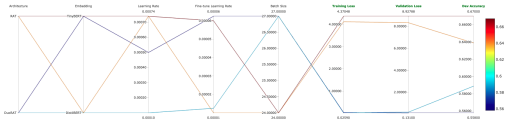


Figure 5. Results of Best Configurations For Each Model