TABERT: Pretraining for Joint Understanding of Textual and Tabular Data

Pengcheng Yin* **Graham Neubig**
Carnegie Mellon University
{pcyin,gneubig}@cs.cmu.edu

Wen-tau Yih **Sebastian Riedel**
Facebook AI Research
{scottyih,sriedel}@fb.com

Abstract

Recent years have witnessed the burgeoning of pretrained language models (LMs) for text-based natural language (NL) understanding tasks. Such models are typically trained on free-form NL text, hence may not be suitable for tasks like semantic parsing over structured data, which require reasoning over both free-form NL questions and structured tabular data (*e.g.*, database tables). In this paper we present TABERT, a pretrained LM that jointly learns representations for NL sentences and (semi-)structured tables. TABERT is trained on a large corpus of 26 million tables and their English contexts. In experiments, neural semantic parsers using TABERT as feature representation layers achieve new best results on the challenging weakly-supervised semantic parsing benchmark WIKITABLEQUESTIONS, while performing competitively on the text-to-SQL dataset SPIDER.[1]

1 Introduction

Recent years have witnessed a rapid advance in the ability to understand and answer questions about free-form natural language (NL) text (Rajpurkar et al., 2016), largely due to large-scale, pretrained language models (LMs) like BERT (Devlin et al., 2019). These models allow us to capture the syntax and semantics of text via representations learned in an unsupervised manner, before fine-tuning the model to downstream tasks (Melamud et al., 2016; McCann et al., 2017; Peters et al., 2018; Liu et al., 2019b; Yang et al., 2019; Goldberg, 2019). It is also relatively easy to apply such pretrained LMs to comprehension tasks that are modeled as text span selection problems, where the boundary of an answer span can be predicted using a simple classifier on top of the LM (Joshi et al., 2019).

However, it is less clear how one could pretrain and fine-tune such models for other QA tasks that involve joint reasoning over both free-form NL text and *structured* data. One example task is semantic parsing for access to databases (DBs) (Zelle and Mooney, 1996; Berant et al., 2013; Yih et al., 2015), the task of transducing an NL utterance (*e.g.*, *"Which country has the largest GDP?"*) into a structured query over DB tables (*e.g.*, SQL querying a database of economics). A key challenge in this scenario is understanding the structured schema of DB tables (*e.g.*, the name, data type, and stored values of columns), and more importantly, the alignment between the input text and the schema (*e.g.*, the token *"GDP"* refers to the `Gross Domestic Product` column), which is essential for inferring the correct DB query (Berant and Liang, 2014).

Neural semantic parsers tailored to this task therefore attempt to learn joint representations of NL utterances and the (semi-)structured schema of DB tables (*e.g.*, representations of its columns or cell values, as in Krishnamurthy et al. (2017); Bogin et al. (2019b); Wang et al. (2019a), *inter alia*). However, this unique setting poses several challenges in applying pretrained LMs. First, information stored in DB tables exhibit strong underlying structure, while existing LMs (*e.g.*, BERT) are solely trained for encoding free-form text. Second, a DB table could potentially have a large number of rows, and naively encoding all of them using a resource-heavy LM is computationally intractable. Finally, unlike most text-based QA tasks (*e.g.*, SQuAD, Rajpurkar et al. (2016)) which could be formulated as a generic answer span selection problem and solved by a pretrained model with additional classification layers, semantic parsing is highly domain-specific, and the architecture of a neural parser is strongly coupled with the structure of its underlying DB (*e.g.*, systems for SQL-based and other types of DBs use different encoder mod-

*Work done while at Facebook AI Research.
[1]Available at github.com/facebookresearch/TaBERT

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 8413–8426
July 5 - 10, 2020.

els). In fact, existing systems have attempted to leverage BERT, but each with their own domain-specific, in-house strategies to encode the structured information in the DB (Guo et al., 2019; Zhang et al., 2019a; Hwang et al., 2019), and importantly, without pretraining representations on structured data. These challenges call for development of general-purpose pretraining approaches tailored to learning representations for both NL utterances and structured DB tables.

In this paper we present TABERT, a pretraining approach for joint understanding of NL text and (semi-)structured tabular data (§ 3). TABERT is built on top of BERT, and jointly learns contextual representations for utterances and the structured schema of DB tables (*e.g.*, a vector for each utterance token and table column). Specifically, TABERT linearizes the structure of tables to be compatible with a Transformer-based BERT model. To cope with large tables, we propose *content snapshots*, a method to encode a subset of table content most relevant to the input utterance. This strategy is further combined with a *vertical attention* mechanism to share information among cell representations in different rows (§ 3.1). To capture the association between tabular data and related NL text, TABERT is pretrained on a parallel corpus of 26 million tables and English paragraphs (§ 3.2).

TABERT can be plugged into a neural semantic parser as a general-purpose encoder to compute representations for utterances and tables. Our key insight is that although semantic parsers are highly domain-specific, most systems rely on representations of input utterances and the table schemas to facilitate subsequent generation of DB queries, and these representations can be provided by TABERT, regardless of the domain of the parsing task.

We apply TABERT to two different semantic parsing paradigms: (1) a classical supervised learning setting on the SPIDER text-to-SQL dataset (Yu et al., 2018c), where TABERT is fine-tuned together with a task-specific parser using parallel NL utterances and labeled DB queries (§ 4.1); and (2) a challenging weakly-supervised learning benchmark WIKITABLEQUESTIONS (Pasupat and Liang, 2015), where a system has to infer latent DB queries from its execution results (§ 4.2). We demonstrate TABERT is effective in both scenarios, showing that it is a drop-in replacement of a parser's original encoder for computing contextual representations of NL utterances and DB tables. Specifically, systems augmented with TABERT outperforms their counterparts using BERT, registering state-of-the-art performance on WIKITABLEQUESTIONS, while performing competitively on SPIDER (§ 5).

2 Background

Semantic Parsing over Tables Semantic parsing tackles the task of translating an NL utterance $\boldsymbol{u}$ into a formal meaning representation (MR) $\boldsymbol{z}$. Specifically, we focus on parsing utterances to access database tables, where $\boldsymbol{z}$ is a structured query (*e.g.*, an SQL query) executable on a set of relational DB tables $\mathcal{T} = \{T_t\}$. A relational table T is a listing of N rows $\{R_i\}_{i=1}^{N}$ of data, with each row R_i consisting of M cells $\{s_{\langle i,j \rangle}\}_{j=1}^{M}$, one for each column c_j. Each cell $s_{\langle i,j \rangle}$ contains a list of tokens.

Depending on the underlying data representation schema used by the DB, a table could either be fully structured with strongly-typed and normalized contents (*e.g.*, a table column named `distance` has a unit of `kilometers`, with all of its cell values, like *200*, bearing the same unit), as is commonly the case for SQL-based DBs (§ 4.1). Alternatively, it could be semi-structured with unnormalized, textual cell values (*e.g.*, *200 km*, § 4.2). The query language could also take a variety of forms, from general-purpose DB access languages like SQL to domain-specific ones tailored to a particular task.

Given an utterance and its associated tables, a neural semantic parser generates a DB query from the vector representations of the utterance tokens and the structured schema of tables. In this paper we refer *schema* as the set of columns in a table, and its *representation* as the list of vectors that represent its columns[2]. We will introduce how TABERT computes these representations in § 3.1.

Masked Language Models Given a sequence of NL tokens $\boldsymbol{x} = x_1, x_2, \ldots, x_n$, a masked language model (*e.g.*, BERT) is an LM trained using the masked language modeling objective, which aims to recover the original tokens in $\boldsymbol{x}$ from a "corrupted" context created by randomly masking out certain tokens in $\boldsymbol{x}$. Specifically, let $\boldsymbol{x}_m = \{x_{i_1}, \ldots, x_{i_m}\}$ be the subset of tokens in $\boldsymbol{x}$ selected to be masked out, and $\widetilde{\boldsymbol{x}}$ denote the masked sequence with tokens in $\boldsymbol{x}_m$ replaced by a `[MASK]` symbol. A masked LM defines a distribu-

[2]Column representations for more complex schemas, *e.g.*, those capturing inter-table dependency via primary and foreign keys, could be derived from these table-wise representations.

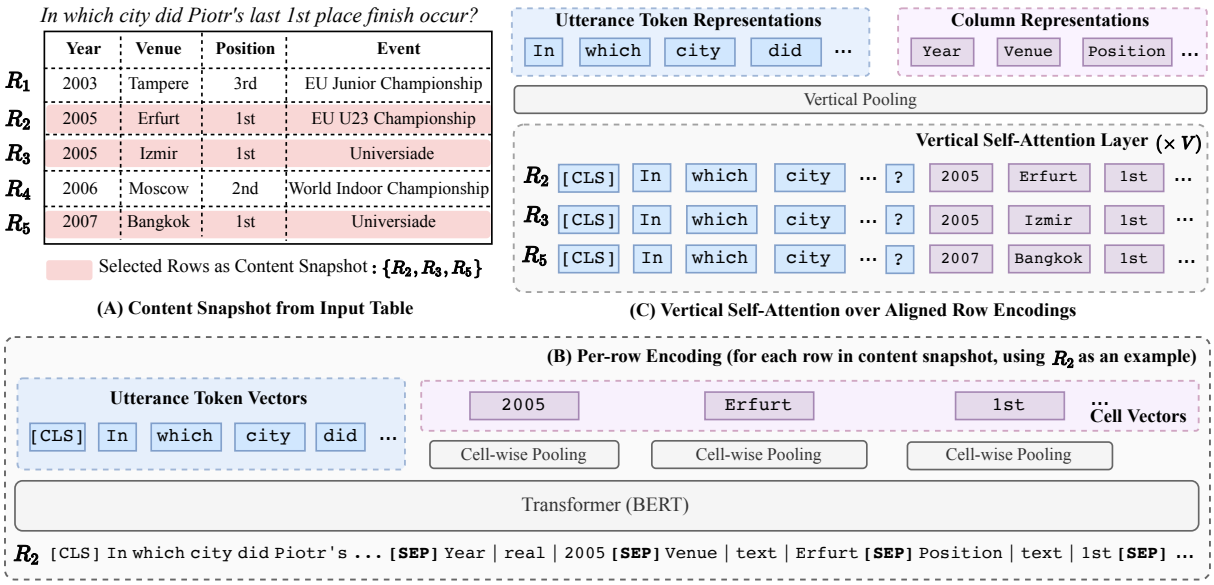

Figure 1: Overview of TABERT for learning representations of utterances and table schemas with an example from WIKITABLEQUESTIONS[3]. **(A)** A content snapshot of the table is created based on the input NL utterance. **(B)** Each row in the snapshot is encoded by a Transformer (only R_2 is shown), producing row-wise encodings for utterance tokens and cells. **(C)** All row-wise encodings are aligned and processed by V vertical self-attention layers, generating utterance and column representations.

tion $p_{\theta}(\boldsymbol{x}_m|\widetilde{\boldsymbol{x}})$ over the target tokens $\boldsymbol{x}_m$ given the masked context $\widetilde{\boldsymbol{x}}$.

BERT parameterizes $p_{\theta}(\boldsymbol{x}_m|\widetilde{\boldsymbol{x}})$ using a Transformer model. During the pretraining phase, BERT maximizes $p_{\theta}(\boldsymbol{x}_m|\widetilde{\boldsymbol{x}})$ on large-scale textual corpora. In the fine-tuning phase, the pretrained model is used as an encoder to compute representations of input NL tokens, and its parameters are jointly tuned with other task-specific neural components.

3 TABERT: Learning Joint Representations over Textual and Tabular Data

We first present how TABERT computes representations for NL utterances and table schemas (§ 3.1), and then describe the pretraining procedure (§ 3.2).

3.1 Computing Representations for NL Utterances and Table Schemas

Fig. 1 presents a schematic overview of TABERT. Given an utterance $\boldsymbol{u}$ and a table T, TABERT first creates a *content snapshot* of T. This snapshot consists of sampled rows that summarize the information in T most relevant to the input utterance. The model then linearizes each row in the snapshot, concatenates each linearized row with the utterance, and uses the concatenated string as input to a Transformer (*e.g.*, BERT) model, which outputs row-wise encoding vectors of utterance tokens and cells. The encodings for all the rows in the snapshot are fed into a series of vertical self-attention layers, where a cell representation (or an utterance token representation) is computed by attending to vertically-aligned vectors of the same column (or the same NL token). Finally, representations for each utterance token and column are generated from a pooling layer.

Content Snapshot One major feature of TABERT is its use of the table *contents*, as opposed to just using the column names, in encoding the table schema. This is motivated by the fact that contents provide more detail about the semantics of a column than just the column's name, which might be ambiguous. For instance, the `Venue` column in Fig. 1 which is used to answer the example question actually refers to *host cities*, and encoding the sampled cell values while creating its representation may help match the term "*city*" in the input utterance to this column.

However, a DB table could potentially have a large number of rows, with only few of them actually relevant to answering the input utterance. Encoding all of the contents using a resource-heavy Transformer is both computationally intractable and likely not necessary. Thus, we instead use a *content snapshot* consisting of only a few rows that are most relevant to the input utterance, providing an efficient approach to calculate content-sensitive column representations from cell values.

We use a simple strategy to create content snap-

[3]Example adapted from `stanford.io/38iZ8Pf`

shots of K rows based on the relevance between the utterance and a row. For $K > 1$, we select the top-K rows in the input table that have the highest n-gram overlap ratio with the utterance.[4] For $K = 1$, to include in the snapshot as much information relevant to the utterance as possible, we create a synthetic row by selecting the cell values from each column that have the highest n-gram overlap with the utterance. Using synthetic rows in this restricted setting is motivated by the fact that cell values most relevant to answer the utterance could come from multiple rows. As an example, consider the utterance *"How many more participants were there in 2008 than in the London Olympics?"*, and an associating table with columns `Year`, `Host City` and `Number of Participants`, the most relevant cells to the utterance, `2008` (from `Year`) and `London` (from `Host City`), are from different rows, which could be included in a single synthetic row. In the initial experiments we found synthetic rows also help stabilize learning.

Row Linearization TABERT creates a linearized sequence for each row in the content snapshot as input to the Transformer model. Fig. 1(B) depicts the linearization for R_2, which consists of a concatenation of the utterance, columns, and their cell values. Specifically, each cell is represented by the name and data type[5] of the column, together with its actual value, separated by a vertical bar. As an example, the cell $s_{\langle 2,1 \rangle}$ valued `2005` in R_2 in Fig. 1 is encoded as

$$\underbrace{\texttt{Year}}_{\text{Column Name}} \mid \underbrace{\texttt{real}}_{\text{Column Type}} \mid \underbrace{\texttt{2005}}_{\text{Cell Value}} \quad (1)$$

The linearization of a row is then formed by concatenating the above string encodings of all the cells, separated by the `[SEP]` symbol. We then prefix the row linearization with utterance tokens as input sequence to the Transformer.

Existing works have applied different linearization strategies to encode tables with Transformers (Hwang et al., 2019; Chen et al., 2019), while our row approach is specifically designed for encoding content snapshots. We present in § 5 results with different linearization choices.

[4]We use $n \leq 3$ in our experiments. Empirically this simple matching heuristic is able to correctly identify the best-matched rows for 40 out of 50 sampled examples on WIKITABLEQUESTIONS.

[5]We use two data types, `text`, and `real` for numbers, predicted by majority voting over the NER labels of cell tokens.

Vertical Self-Attention Mechanism The base Transformer model in TABERT outputs vector encodings of utterance and cell tokens for each row. These row-level vectors are computed separately and therefore independent of each other. To allow for information flow across cell representations of different rows, we propose vertical self-attention, a self-attention mechanism that operates over vertically aligned vectors from different rows.

As in Fig. 1(C), TABERT has V stacked vertical-level self-attention layers. To generate aligned inputs for vertical attention, we first compute a fixed-length initial vector for each cell at position $\langle i, j \rangle$, which is given by mean-pooling over the sequence of the Transformer's output vectors that correspond to its variable-length linearization as in Eq. (1). Next, the sequence of word vectors for the NL utterance (from the base Transformer model) are concatenated with the cell vectors as initial inputs to the vertical attention layer.

Each vertical attention layer has the same parameterization as the Transformer layer in (Vaswani et al., 2017), but operates on vertically aligned elements, *i.e.*, utterance and cell vectors that correspond to the same question token and column, respectively. This vertical self-attention mechanism enables the model to aggregate information from different rows in the content snapshot, allowing TABERT to capture cross-row dependencies on cell values.

Utterance and Column Representations A representation $\mathbf{c}_j$ is computed for each column c_j by mean-pooling over its vertically aligned cell vectors, $\{\mathbf{s}_{\langle i,j \rangle} : R_i \text{ in content snapshot}\}$, from the last vertical layer. A representation for each utterance token, $\mathbf{x}_j$, is computed similarly over the vertically aligned token vectors. These representations will be used by downstream neural semantic parsers. TABERT also outputs an optional fixed-length table representation $\mathbf{T}$ using the representation of the prefixed `[CLS]` symbol, which is useful for parsers that operate on multiple DB tables.

3.2 Pretraining Procedure

Training Data Since there is no large-scale, high-quality parallel corpus of NL text and structured tables, we instead use semi-structured tables that commonly exist on the Web as a surrogate data source. As a first step in this line, we focus on collecting parallel data in English, while extending to multilingual scenarios would be an

interesting avenue for future work. Specifically, we collect tables and their surrounding NL text from English Wikipedia and the WDC WebTable Corpus (Lehmberg et al., 2016), a large-scale table collection from CommonCrawl. The raw data is extremely noisy, and we apply aggressive cleaning heuristics to filter out invalid examples (*e.g.*, examples with HTML snippets or in foreign languages, and non-relational tables without headers). See Appendix § A.1 for details of data pre-processing. The pre-processed corpus contains 26.6 million parallel examples of tables and NL sentences. We perform sub-tokenization using the Wordpiece tokenizer shipped with BERT.

Unsupervised Learning Objectives We apply different objectives for learning representations of the NL context and structured tables. For NL contexts, we use the standard Masked Language Modeling (MLM) objective (Devlin et al., 2019), with a masking rate of 15% sub-tokens in an NL context.

For learning column representations, we design two objectives motivated by the intuition that a column representation should contain both the general information of the column (*e.g.*, its name and data type), and representative cell values relevant to the NL context. First, a **Masked Column Prediction (MCP)** objective encourages the model to recover the names and data types of masked columns. Specifically, we randomly select 20% of the columns in an input table, masking their names and data types in each row linearization (*e.g.*, if the column `Year` in Fig. 1 is selected, the tokens `Year` and `real` in Eq. (1) will be masked). Given the column representation $\mathbf{c}_j$, TABERT is trained to predict the bag of masked (name and type) tokens from $\mathbf{c}_j$ using a multi-label classification objective. Intuitively, MCP encourages the model to recover column information from its contexts.

Next, we use an auxiliary **Cell Value Recovery (CVR)** objective to ensure information of representative cell values in content snapshots is retained after additional layers of vertical self-attention. Specifically, for each masked column c_j in the above MCP objective, CVR predicts the original tokens of each cell $s_{\langle i,j \rangle}$ (of c_j) in the content snapshot conditioned on its cell vector $\mathbf{s}_{\langle i,j \rangle}$.[6] For instance, for the example cell $s_{\langle 2,1 \rangle}$ in Eq. (1), we predict its value `2005` from $\mathbf{s}_{\langle 2,1 \rangle}$. Since a cell could have multiple value tokens, we apply the span-based prediction objective (Joshi et al., 2019). Specifically, to predict a cell token $s_{\langle i,j \rangle_k} \in s_{\langle i,j \rangle}$, its positional embedding $\mathbf{e}_k$ and the cell representations $\mathbf{s}_{\langle i,j \rangle}$ are fed into a two-layer network $f(\cdot)$ with GeLU activations (Hendrycks and Gimpel, 2016). The output of $f(\cdot)$ is then used to predict the original value token $s_{\langle i,j \rangle_k}$ from a softmax layer.

[6]The cell value tokens are not masked in the input sequence, since predicting masked cell values is challenging even with the presence of its surrounding context.

4 Example Application: Semantic Parsing over Tables

We apply TABERT for representation learning on two semantic parsing paradigms, a classical supervised text-to-SQL task over structured DBs (§ 4.1), and a weakly supervised parsing problem on semi-structured Web tables (§ 4.2).

4.1 Supervised Semantic Parsing

Benchmark Dataset Supervised learning is the typical scenario of learning a parser using parallel data of utterances and queries. We use SPIDER (Yu et al., 2018c), a text-to-SQL dataset with 10,181 examples across 200 DBs. Each example consists of an utterance (*e.g.*, *"What is the total number of languages used in Aruba?"*), a DB with one or more tables, and an annotated SQL query, which typically involves joining multiple tables to get the answer (*e.g.*, `SELECT COUNT(*) FROM Country JOIN Lang ON Country.Code = Lang.CountryCode WHERE Name = 'Aruba'`).

Base Semantic Parser We aim to show TABERT could help improve upon an already strong parser. Unfortunately, at the time of writing, none of the top systems on SPIDER were publicly available. To establish a reasonable testbed, we developed our in-house system based on TranX (Yin and Neubig, 2018), an open-source general-purpose semantic parser. TranX translates an NL utterance into an intermediate meaning representation guided by a user-defined grammar. The generated intermediate MR could then be deterministically converted to domain-specific query languages (*e.g.*, SQL).

We use TABERT as encoder of utterances and table schemas. Specifically, for a given utterance $\boldsymbol{u}$ and a DB with a set of tables $\mathcal{T} = \{T_t\}$, we first pair $\boldsymbol{u}$ with each table T_t in $\mathcal{T}$ as inputs to TABERT, which generates $|\mathcal{T}|$ sets of table-specific representations of utterances and columns. At each time step, an LSTM decoder performs hierarchical attention (Libovický and Helcl, 2017) over the list of table-specific representations, constructing an

MR based on the predefined grammar. Following the IRNet model (Guo et al., 2019) which achieved the best performance on SPIDER as the time of writing, we use SemQL, a simplified version of the SQL, as the underlying grammar. We refer interested readers to Appendix § B.1 for details of our system.

4.2 Weakly Supervised Semantic Parsing

Benchmark Dataset Weakly supervised semantic parsing considers the reinforcement learning task of inferring the correct query from its execution results (*i.e.*, whether the answer is correct). Compared to supervised learning, weakly supervised parsing is significantly more challenging, as the parser does not have access to the labeled query, and has to explore the exponentially large search space of possible queries guided by the noisy binary reward signal of execution results.

WIKITABLEQUESTIONS (Pasupat and Liang, 2015) is a popular dataset for weakly supervised semantic parsing, which has 22,033 utterances and 2,108 semi-structured Web tables from Wikipedia.[7] Compared to SPIDER, examples in this dataset do not involve joining multiple tables, but typically require compositional, multi-hop reasoning over a series of entries in the given table (*e.g.*, to answer the example in Fig. 1 the parser needs to reason over the row set $\{R_2, R_3, R_5\}$, locating the `Venue` field with the largest value of `Year`).

Base Semantic Parser MAPO (Liang et al., 2018) is a strong system for weakly supervised semantic parsing. It improves the sample efficiency of the REINFORCE algorithm by biasing the exploration of queries towards the high-rewarding ones already discovered by the model. MAPO uses a domain-specific query language tailored to answering compositional questions on single tables, and its utterances and column representations are derived from an LSTM encoder, which we replaced with our TABERT model. See Appendix § B.2 for details of MAPO and our adaptation.

5 Experiments

In this section we evaluate TABERT on downstream tasks of semantic parsing to DB tables.

[7] While some of the 421 testing Wikipedia tables might be included in our pretraining corpora, they only account for a very tiny fraction. In our pilot study, we also found pretraining only on Wikipedia tables resulted in worse performance.

Pretraining Configuration We train two variants of the model, $\text{TABERT}_{\texttt{Base}}$ and $\text{TABERT}_{\texttt{Large}}$, with the underlying Transformer model initialized with the uncased versions of $\text{BERT}_{\texttt{Base}}$ and $\text{BERT}_{\texttt{Large}}$, respectively.[8] During pretraining, for each table and its associated NL context in the corpus, we create a series of training instances of paired NL sentences (as synthetically generated utterances) and tables (as content snapshots) by (1) sliding a (non-overlapping) context window of sentences with a maximum length of 128 tokens, and (2) using the NL tokens in the window as the utterance, and pairing it with randomly sampled rows from the table as content snapshots. TABERT is implemented in PyTorch using distributed training. Refer to Appendix § A.2 for details of pretraining.

Comparing Models We mainly present results for two variants of TABERT by varying the size of content snapshots K. **TABERT**$(\mathbf{K = 3})$ uses three rows from input tables as content snapshots and three vertical self-attention layers. **TABERT**$(\mathbf{K = 1})$ uses one synthetically generated row as the content snapshot as described in § 3.1. Since this model does not have multi-row input, we do not use additional vertical attention layers (and the cell value recovery learning objective). Its column representation $\mathbf{c}_j$ is defined by mean-pooling over the Transformer's output encodings that correspond to the column name (*e.g.*, the representation for the `Year` column in Fig. 1 is derived from the vector of the `Year` token in Eq. (1)). We find this strategy gives better results compared with using the cell representation $\mathbf{s}_j$ as $\mathbf{c}_j$. We also compare with **BERT** using the same row linearization and content snapshot approach as TABERT$(\text{K} = 1)$, which reduces to a TABERT$(\text{K} = 1)$ model without pretraining on tabular corpora.

Evaluation Metrics As standard, we report execution accuracy on WIKITABLEQUESTIONS and exact-match accuracy of DB queries on SPIDER.

5.1 Main Results

Tab. 1 and Tab. 2 summarize the end-to-end evaluation results on WIKITABLEQUESTIONS and SPIDER, respectively. First, comparing with existing strong semantic parsing systems, we found our

[8] We also attempted to train TABERT on our collected corpus from scratch without initialization from BERT, but with inferior results, potentially due to the average lower quality of web-scraped tables compared to purely textual corpora. We leave improving the quality of training data as future work.

Previous Systems on WikiTableQuestions		
Model	DEV	TEST
Pasupat and Liang (2015)	37.0	37.1
Neelakantan et al. (2016)	34.1	34.2
Ensemble 15 Models	37.5	37.7
Zhang et al. (2017)	40.6	43.7
Dasigi et al. (2019)	43.1	44.3
Agarwal et al. (2019)	43.2	44.1
Ensemble 10 Models	–	46.9
Wang et al. (2019b)	43.7	44.5

Our System based on MAPO (Liang et al., 2018)				
	DEV	Best	TEST	Best
Base Parser†	42.3 ±0.3	42.7	43.1 ±0.5	43.8
w/ $\text{BERT}_{\texttt{Base}}$ (K = 1)	49.6 ±0.5	50.4	49.4 ±0.5	49.2
– content snapshot	49.1 ±0.6	50.0	48.8 ±0.9	50.2
w/ $\text{TaBERT}_{\texttt{Base}}$ (K = 1)	51.2 ±0.5	51.6	50.4 ±0.5	51.2
– content snapshot	49.9 ±0.4	50.3	49.4 ±0.4	50.0
w/ $\text{TaBERT}_{\texttt{Base}}$ (K = 3)	51.6 ±0.5	52.4	51.4 ±0.3	51.3
w/ $\text{BERT}_{\texttt{Large}}$ (K = 1)	50.3 ±0.4	50.8	49.6 ±0.5	50.1
w/ $\text{TaBERT}_{\texttt{Large}}$ (K = 1)	51.6 ±1.1	52.7	51.2 ±0.9	51.5
w/ $\text{TaBERT}_{\texttt{Large}}$ (K = 3)	**52.2** ±0.7	**53.0**	**51.8** ±0.6	**52.3**

Table 1: Execution accuracies on WIKITABLEQUESTIONS. †Results from Liang et al. (2018). (TA)BERT models are evaluated with 10 random runs. We report mean, standard deviation and the best results. TEST↦BEST refers to the result from the run with the best performance on DEV. set.

Top-ranked Systems on Spider Leaderboard	
Model	DEV. ACC.
Global–GNN (Bogin et al., 2019a)	52.7
EditSQL + BERT (Zhang et al., 2019a)	57.6
RatSQL (Wang et al., 2019a)	60.9
IRNet + BERT (Guo et al., 2019)	60.3
+ Memory + Coarse-to-Fine	61.9
IRNet V2 + BERT	63.9
RyanSQL + BERT (Choi et al., 2020)	**66.6**

Our System based on TranX (Yin and Neubig, 2018)		
	Mean	Best
w/ $\text{BERT}_{\texttt{Base}}$ (K = 1)	61.8 ±0.8	62.4
– content snapshot	59.6 ±0.7	60.3
w/ $\text{TaBERT}_{\texttt{Base}}$ (K = 1)	63.3 ±0.6	64.2
– content snapshot	60.4 ±1.3	61.8
w/ $\text{TaBERT}_{\texttt{Base}}$ (K = 3)	63.3 ±0.7	64.1
w/ $\text{BERT}_{\texttt{Large}}$ (K = 1)	61.3 ±1.2	62.9
w/ $\text{TaBERT}_{\texttt{Large}}$ (K = 1)	64.0 ±0.4	64.4
w/ $\text{TaBERT}_{\texttt{Large}}$ (K = 3)	**64.5** ±0.6	**65.2**

Table 2: Exact match accuracies on the public development set of SPIDER. Models are evaluated with 5 random runs.

parsers with TABERT as the utterance and table encoder perform competitively. On the test set of WIKITABLEQUESTIONS, MAPO augmented with a $\text{TaBERT}_{\texttt{Large}}$ model with three-row content snapshots, $\text{TaBERT}_{\texttt{Large}}$(K = 3), registers a single-model exact-match accuracy of 52.3%, surpassing the previously best ensemble system (46.9%) from Agarwal et al. (2019) by 5.4% absolute.

On SPIDER, our semantic parser based on TranX and SemQL (§ 4.1) is conceptually similar to the base version of IRNet as both systems use the SemQL grammar, while our system has a simpler decoder. Interestingly, we observe that its performance with $\text{BERT}_{\texttt{Base}}$ (61.8%) matches the full BERT-augmented IRNet model with a stronger decoder using augmented memory and coarse-to-fine decoding (61.9%). This confirms that our base parser is an effective baseline. Augmented with representations produced by $\text{TaBERT}_{\texttt{Large}}$(K = 3), our parser achieves up to 65.2% exact-match accuracy, a 2.8% increase over the base model using $\text{BERT}_{\texttt{Base}}$. Note that while other competitive systems on the leaderboard use BERT with more sophisticated semantic parsing models, our best DEV. result is already close to the score registered by the best submission (RyanSQL+BERT). This suggests that if they instead used TABERT as the representation layer, they would see further gains.

Comparing semantic parsers augmented with TABERT and BERT, we found TABERT is more effective across the board. We hypothesize that the performance improvements would be attributed by two factors. First, pre-training on large parallel textual and tabular corpora helps TABERT learn to encode structure-rich tabular inputs in their linearized form (Eq. (1)), whose format is different from the ordinary natural language data that BERT is trained on. Second, pre-training on parallel data could also helps the model produce representations that better capture the alignment between an utterance and the relevant information presented in the structured schema, which is important for semantic parsing.

Overall, the results on the two benchmarks demonstrate that pretraining on aligned textual and tabular data is necessary for joint understanding of NL utterances and tables, and TABERT works well with both structured (SPIDER) and semi-structured (WIKITABLEQUESTIONS) DBs, and agnostic of the task-specific structures of semantic parsers.

Effect of Content Snapshots In this paper we propose using content snapshots to capture the information in input DB tables that is most relevant to answering the NL utterance. We therefore study the effectiveness of including content snapshots when generating schema representations. We include in Tab. 1 and Tab. 2 results of models without using content in row linearization ("−content snapshot"). Under this setting a column is rep-

u: *How many years before was the film Bacchae out before the Watermelon?*

Input to TABERT$_{\text{Large}}$ (K = 3) ▷ *Content Snapshot with Three Rows*

Film	Year	Function	Notes
The Bacchae	2002	Producer	Screen adaptation of...
The Trojan Women	2004	Producer/Actress	Documutary film...
The Watermelon	2008	Producer	Oddball romantic comedy...

Input to TABERT$_{\text{Large}}$ (K = 1) ▷ *Content Snapshot with One Synthetic Row*

Film	Year	Function	Notes
The Watermelon	2013	Producer	Screen adaptation of...

Table 3: Content snapshots generated by two models for a WIKITABLEQUESTIONS DEV. example. Matched tokens between the question and content snapshots are underlined.

resented as "`Column Name | Type`" without cell values (*c.f.*, Eq. (1)). We find that content snapshots are helpful for both BERT and TABERT, especially for TABERT. As discussed in § 3.1, encoding sampled values from columns in learning their representations helps the model infer alignments between entity and relational phrases in the utterance and the corresponding column. This is particularly helpful for identifying relevant columns from a DB table that is mentioned in the input utterance. As an example, empirically we observe that on SPIDER our semantic parser with TABERT$_{\text{Base}}$ using just one row of content snapshots (K = 1) registers a higher accuracy of selecting the correct columns when generating SQL queries (*e.g.*, columns in `SELECT` and `WHERE` clauses), compared to the TABERT$_{\text{Base}}$ model without encoding content information (87.4% v.s. 86.4%).

Additionally, comparing TABERT using one synthetic row (K = 1) and three rows from input tables (K = 3) as content snapshots, the latter generally performs better. Intuitively, encoding more table contents relevant to the input utterance could potentially help answer questions that involve reasoning over information across multiple rows in the table. Tab. 3 shows such an example, and to answer this question a parser need to subtract the values of `Year` in the rows for *"The Watermelon"* and *"The Bacchae"*. TABERT$_{\text{Large}}$ (K = 3) is able to capture the two target rows in its content snapshot and generates the correct DB query, while the TABERT$_{\text{Large}}$(K = 1) model with only one row as content snapshot fails to answer this example.

Effect of Row Linearization TABERT uses row linearization to represent a table row as sequential input to Transformer. Tab. 4 (upper half) presents results using various linearization methods. We find adding type information and content snapshots improves performance, as they provide more hints about the meaning of a column.

Cell Linearization Template	WIKIQ.	SPIDER
Pretrained TABERT$_{\text{Base}}$ Models (K = 1)		
`Column Name`	49.6 ±0.4	60.0 ±1.1
`Column Name | Type`† (−content snap.)	49.9 ±0.4	60.4 ±1.3
`Column Name | Type | Cell Value`†	51.2 ±0.5	63.3 ±0.6
BERT$_{\text{Base}}$ Models		
`Column Name` (Hwang et al., 2019)	49.0 ±0.4	58.6 ±0.3
`Column Name is Cell Value` (Chen19)	50.2 ±0.4	63.1 ±0.7

Table 4: Performance of pretrained TABERT$_{\text{Base}}$ models and BERT$_{\text{Base}}$ on the DEV. sets with different linearization methods. Slot names are underlined. †Results copied from Tab. 1 and Tab. 2.

Learning Objective	WIKIQ.	SPIDER
MCP only	51.6 ±0.7	62.6 ±0.7
MCP + CVR	51.6 ±0.5	63.3 ±0.7

Table 5: Performance of pretrained TABERT$_{\text{Base}}$(K = 3) on DEV. sets with different pretraining objectives.

We also compare with existing linearization methods in literature using a TABERT$_{\text{Base}}$ model, with results shown in Tab. 4 (lower half). Hwang et al. (2019) uses BERT to encode concatenated column names to learn column representations. In line with our previous discussion on the effectiveness content snapshots, this simple strategy without encoding cell contents underperforms (although with TABERT$_{\text{Base}}$ pretrained on our tabular corpus the results become slightly better). Additionally, we remark that linearizing table contents has also be applied to other BERT-based tabular reasoning tasks. For instance, Chen et al. (2019) propose a "natural" linearization approach for checking if an NL statement entails the factual information listed in a table using a binary classifier with representations from BERT, where a table is linearized by concatenating the semicolon-separated cell linearization for all rows. Each cell is represented by a phrase "`column name is cell value`". For completeness, we also tested this cell linearization approach, and find BERT$_{\text{Base}}$ achieved improved results. We leave pretraining TABERT with this linearization strategy as promising future work.

Impact of Pretraining Objectives TABERT uses two objectives (§ 3.2), a masked column prediction (MCP) and a cell value recovery (CVR) objective, to learn column representations that could capture both the general information of the column (via MCP) and its representative cell values related to the utterance (via CVR). Tab. 5 shows ablation results of pretraining TABERT with different objectives. We find TABERT trained with both MCP and the auxiliary CVR objectives gets a slight advantage, suggesting CVR could potentially lead to

more representative column representations with additional cell information.

6 Related Works

Semantic Parsing over Tables Tables are important media of world knowledge. Semantic parsers have been adapted to operate over structured DB tables (Wang et al., 2015; Xu et al., 2017; Dong and Lapata, 2018; Yu et al., 2018b; Shi et al., 2018; Wang et al., 2018), and open-domain, semi-structured Web tables (Pasupat and Liang, 2015; Sun et al., 2016; Neelakantan et al., 2016). To improve representations of utterances and tables for neural semantic parsing, existing systems have applied pretrained word embeddings (*e.g.*., GloVe, as in Zhong et al. (2017); Yu et al. (2018a); Sun et al. (2018); Liang et al. (2018)), and BERT-family models for learning joint contextual representations of utterances and tables, but with domain-specific approaches to encode the structured information in tables (Hwang et al., 2019; He et al., 2019; Guo et al., 2019; Zhang et al., 2019a). TABERT advances this line of research by presenting a general-purpose, pretrained encoder over parallel corpora of Web tables and NL context. Another relevant direction is to augment representations of columns from an individual table with global information of its linked tables defined by the DB schema (Bogin et al., 2019a; Wang et al., 2019a). TABERT could also potentially improve performance of these systems with improved table-level representations.

Knowledge-enhanced Pretraining Recent pretraining models have incorporated structured information from knowledge bases (KBs) or other structured semantic annotations into training contextual word representations, either by fusing vector representations of entities and relations on KBs into word representations of LMs (Peters et al., 2019; Zhang et al., 2019b,c), or by encouraging the LM to recover KB entities and relations from text (Sun et al., 2019; Liu et al., 2019a). TABERT is broadly relevant to this line in that it also exposes an LM with structured data (*i.e.*, tables), while aiming to learn joint representations for both textual and structured tabular data.

7 Conclusion and Future Work

We present TABERT, a pretrained encoder for joint understanding of textual and tabular data. We show that semantic parsers using TABERT as a general-purpose feature representation layer achieved strong results on two benchmarks. This work also opens up several avenues for future work. First, we plan to evaluate TABERT on other related tasks involving joint reasoning over textual and tabular data (*e.g.*, table retrieval and table-to-text generation). Second, following the discussions in § 5, we will explore other table linearization strategies with Transformers, improving the quality of pretraining corpora, as well as novel unsupervised objectives. Finally, to extend TABERT to cross-lingual settings with utterances in foreign languages and structured schemas defined in English, we plan to apply more advanced semantic similarity metrics for creating content snapshots.

References

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to generalize from sparse and underspecified rewards. In *ICML*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of ACL*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019a. Global reasoning over database structures for text-to-sql parsing. *ArXiv*, abs/1908.11214.

Ben Bogin, Matthew Gardner, and Jonathan Berant. 2019b. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of ACL*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. TabFact: A large-scale dataset for table-based fact verification. *ArXiv*, abs/1909.02164.

Donghyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *ArXiv*, abs/2004.03125.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke S. Zettlemoyer, and Eduard H. Hovy. 2019. Iterative search for weakly supervised semantic parsing. In *Proceedings of NAACL-HLT*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Li Dong and Mirella Lapata. 2018. coarse-to-fine decoding for neural semantic parsing. In *Proceedings of ACL*.

Yoav Goldberg. 2019. Assessing bert's syntactic abilities. *ArXiv*, abs/1901.05287.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of ACL*.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-sql: reinforce schema representation with context. *ArXiv*, abs/1908.08113.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *ArXiv*, abs/1606.08415.

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

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke S. Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. In *Proceedings of EMNLP*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of EMNLP*.

Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of WWW*.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Proceedings of NIPS*.

Jindrich Libovický and Jindrich Helcl. 2017. Attention strategies for multi-source sequence-to-sequence learning. In *Proceedings of ACL*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2019a. K-bert: Enabling language representation with knowledge graph. *ArXiv*, abs/1909.07606.

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

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Proceedings of NIPS*.

Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of CoNLL*.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *Proceedings of ICLR*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of ACL*.

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

Matthew E. Peters, Mark Neumann, IV RobertLLogan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of EMNLP*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of EMNLP*.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *ArXiv*, abs/1809.05054.

Huan Sun, Hao Ma, Xiaodong He, Wen tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of WWW*.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware SQL generation. In *Proceedings of EMNLP*.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019. Ernie: Enhanced representation through knowledge integration. *ArXiv*, abs/1904.09223.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NIPS*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Margot Richardson. 2019a. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *ArXiv*, abs/1911.04942.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019b. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *EMNLP/IJCNLP*.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Xin Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *ArXiv*, abs/1807.03100.

Daniel C. Wang, Andrew W. Appel, Jeffrey L. Korn, and Christopher S. Serra. 1997. The Zephyr abstract syntax description language. In *Proceedings of DSL*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of ACL*.

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

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Proceedings of NIPS*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of ACL*.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of EMNLP Demonstration Track*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir R. Radev. 2018a. TypeSQL: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of NAACL-HLT*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of EMNLP*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of EMNLP*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of AAAI*.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir R. Radev. 2019a. Editing-based sql query generation for cross-domain context-dependent questions. *ArXiv*, abs/1909.00786.

Yuchen Zhang, Panupong Pasupat, and Percy Liang. 2017. Macro grammars and holistic triggering for efficient semantic parsing. In *Proceedings of EMNLP*.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019b. Ernie: Enhanced language representation with informative entities. In *Proceedings of ACL*.

Zhuosheng Zhang, Yu-Wei Wu, Hai Zhao, Zuchao Li, Shuailiang Zhang, Xi Zhou, and Xiaodong Zhou. 2019c. Semantics-aware bert for language understanding. *ArXiv*, abs/1909.02209.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv*, abs/1709.00103.

Supplementary Materials

A Pretraining Details

A.1 Training Data

We collect parallel examples of tables and their surrounding NL sentences from two sources:

Wikipedia Tables We extract all the tables on English Wikipedia[9]. For each table, we use the preceding three paragraphs as the NL context, as we observe that most Wiki tables are located after where they are described in the body text.

WDC WebTable Corpus (Lehmberg et al., 2016) is a large collection of Web tables extracted from the Common Crawl Web scrape[10]. We use its 2015 English-language relational subset, which consists of 50.8 million relational tables and their surrounding NL contexts.

Preprocessing Our dataset is collected from arbitrary Web tables, which are extremely noisy. We develop a set of heuristics to clean the data by: (1) removing columns whose names have more than 10 tokens; (2) filtering cells with more than two non-ASCII characters or 20 tokens; (3) removing empty or repetitive rows and columns; (4) filtering tables with less than three rows and four columns, and (5) running `spaCy` to identify the data type of columns (text or real value) by majority voting over the NER labels of column tokens, (6) rotating vertically oriented tables. We sub-tokenize the corpus using the Wordpiece tokenizer in Devlin et al. (2019). The pre-processing results in 1.3 million tables from Wikipedia and 25.3 million tables from the WDC corpus.

A.2 Pretraining Setup

As discussed in § 5, we create training instances of NL sentences (as synthetic utterances) and content snapshots from tables by sampling from the parallel corpus of NL contexts and tables. Each epoch contains $37.6M$ training instances. We train TABERT for 10 epochs. Tab. 6 lists the hyper-parameters used in training. Learning rates are validated on the development set of WIKITABLEQUESTIONS. We use a batch size of 512 for large models to reduce training time. The training objective is sum of the three pretraining objectives in § 3.2: the masked language modeling (MLM) objective for utterance tokens, the masked column prediction (MCP) objective for columns, and the column value recovery (CVR) objective for their cell values. An exception is pretraining the TABERT($\mathrm{K} = 1$) models. Since there are no additional vertical attention layers, we do not use the CVR objective, and the MCP objective reduces to the vanilla MLM objective over encodings from the base Transformer model. Our largest model $\text{TABERT}_{\texttt{Large}}(\mathrm{K} = 3)$ takes six days to train for 10 epochs on 128 Tesla V100 GPUs using mixed precision training.

[9] We do not use infoboxes (tables on the top-right of a Wiki page that describe properties of the main topic), as they are not relational tables.

[10] http://webdatacommons.org/webtables

B Semantic Parsers

B.1 Supervised Parsing on SPIDER

Model We develop our text-to-SQL parser based on TranX (Yin and Neubig, 2018), which translates an NL utterance into a tree-structured abstract meaning representation following user-specified grammar, before deterministically convert the generated abstract MR into an SQL query. TranX models the construction process of an abstract MR (tree-structured representation of an SQL query) using a transition-based system, which decomposes its generation story into a sequence of actions following the user defined grammar.

Formally, given an input NL utterance $\boldsymbol{u}$ and a database with a set of tables $\mathcal{T} = \{T_i\}$, the probability of generating of an SQL query (*i.e.*, its semantically equivalent MR) $\boldsymbol{z}$ is decomposed as the production of action probabilities:

$$p(\boldsymbol{z}|\boldsymbol{u}, \mathcal{T}) = \prod p(a_t|a_{<t}, \boldsymbol{u}, \mathcal{T}) \qquad (2)$$

where a_t is the action applied to the hypothesis at time stamp t. $a_{<t}$ denote the previous action history. We refer readers to Yin and Neubig (2018) for details of the transition system and how individual action probabilities are computed. In our adaptation of TranX to text-to-SQL parsing on SPIDER, we follow Guo et al. (2019) and use SemQL as the underlying grammar, which is a simplification of the SQL language. Fig. 2 lists the SemSQL grammar specified using the abstract syntax description language (Wang et al., 1997). Intuitively, the generation starts from a tree-structured derivation with the root production rule `select_stmt`↦`SelectStatement`, which lays out overall the structure of an SQL query. At each time step, the decoder algorithm locates the current opening node on the derivation tree, following a depth-first, left-to-right order. If the opening node

Parameter	TaBERT$_{\text{Base}}$(K = 1)	TaBERT$_{\text{Large}}$(K = 1)	TaBERT$_{\text{Base}}$(K = 3)	TaBERT$_{\text{Large}}$(K = 3)
Batch Size	256	512	512	512
Learning Rate	2×10^{-5}	2×10^{-5}	4×10^{-5}	4×10^{-5}
Max Epoch	10			
Weight Decay	0.01			
Gradient Norm Clipping	1.0			

Table 6: Hyper-parameters using in pretraining

is not a leaf node, the decoder invokes an action a_t which expands the opening node using a production rule with appropriate type. If the current opening node is a leaf node (*e.g.*, a node denoting string literal), the decoder fills in the leaf node using actions that emit terminal values.

To use such a transition system to generate SQL queries, we extend its action space with two new types of actions, SELECTTABLE(T_i) for node of type `table_ref` in Fig. 2, which selects a table T_i (*e.g.*, for predicting target tables for a `FROM` clause), and SELECTCOLUMN(T_i, c_j) for node of type `column_ref`, which selects the column c_j from table T_i (*e.g.*, for predicting a result column used in the `SELECT` clause).

As described in § 4.1, TABERT produces a list of entries, with one entry $\langle \mathbf{T}_i, \mathbf{X}_i, \mathbf{C}_i \rangle$ for each table T_i:

$$\mathbb{M} = \Big\{ \langle \mathbf{T}_i, \mathbf{X}_i = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}, \mathbf{C}_i = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \}\rangle_i \Big\}_{i=1}^{|\mathcal{T}|} \quad (3)$$

where each entry $\langle \mathbf{T}_i, \mathbf{X}_i, \mathbf{C}_i \rangle$ in $\mathbb{M}$ consists of $\mathbf{T}_i$, the representation of table T_i given by the output vector of the prefixed `[CLS]` symbol, the table-specific representations of utterance tokens $\mathbf{X}_i = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$, and representations of columns in T_i, $\mathbf{C}_i = \{\mathbf{c}_1, \mathbf{c}_2, \ldots\}$. At each time step t, the decoder in TranX performs hierarchical attention over representations in $\mathbb{M}$ to compute a context vector. First, a table-wise attention score is computed using the LSTM's previous state, $\mathbf{state}_{t-1}$ with the set of table representations.

$$\text{score}(T_i) = \text{Softmax}\Big(\text{DotProduct}(\mathbf{state}_{t-1}, \text{key}(\mathbf{T}_i)) \Big), \quad (4)$$

where the linear projection $\text{key}(\cdot) \in \mathbb{R}^{256}$ projects the table representations to key space. Next, for each table $T_i \in \mathcal{T}$, a table-wise context vector $\mathbf{ctx}(T_i)$ is generated by attending over the union of vectors in utterance token representations $\mathbf{X}_i$ and column representations $\mathbf{C}_i$:

$$\mathbf{ctx}(T_i) = \text{DotProductAttention}\Big(\mathbf{state}_{t-1}, \text{key}(\mathbf{X}_i \cup \mathbf{C}_i), \text{value}(\mathbf{X}_i \cup \mathbf{C}_i) \Big), \quad (5)$$

with the LSTM state as the query, $\text{key}(\cdot)$ as the key, and another linear transformation $\text{value}(\cdot) \in \mathbb{R}^{256}$ to project the representations to value vectors. The final context vector is then given by the weighted sum of these table-wise context vectors $\mathbf{ctx}(T_i)$ ($i \in \{1, \ldots, |\mathcal{T}|\}$) weighted by the attention scores $\text{score}(T_i)$. The generated context vector is then used to update the state of the decoder LSTM to $\mathbf{state}_t$.

The updated decoder state is then used to compute the probability of carrying out the action defined at time step t, a_t. For a SELECTTABLE(T_i) action, its probability of is defined similarly as Eq. (4). For a SELECTCOLUMN(T_i, c_j) action, it is factorized as the probability of selecting the table T_i (given by Eq. (4)), times the probability of selecting the column c_j. The latter is defined as

$$\text{score}(c_j) = \text{Softmax}\Big(\text{DotProduct}(\mathbf{state}_t, \mathbf{c}_j) \Big). \quad (6)$$

We also add simple entity linking features to the representations in $\mathbb{M}$, defined by the following heuristics: (1) If an utterance token $x \in \boldsymbol{u}$ matches with the name of a table T, we concatenate a trainable embedding vector ($\mathbf{table\_match} \in \mathbb{R}^{16}$) to the representations of x and T. (2) Similarly, we concatenate an embedding vector ($\mathbf{column\_match} \in \mathbb{R}^{16}$) to the representations of an utterance token and a column if their names match. (3) Finally, we concatenate a zero-vector ($\mathbf{0} \in \mathbb{R}^{16}$) to representations of all unmatched elements.

Configuration We use the default configuration of TranX. For TABERT parameters, we use an Adam optimizer with a learning rate of $3e-5$ and linearly decayed learning rate schedule, and

```
select_stmt = SelectStatement(
    distinct distinct,                          # DISTINCT keyword
    expr* result_columns,                       # Columns in SELECT clause
    expr? where_clause,                         # WHERE clause
    order_by_clause? order_by_clause,           # ORDER BY clause
    int? limit_value,                           # LIMIT clause
    table_ref* join_with_tables,                # Tables in the JOIN clause
    compound_stmt? compound_statement           # Compound statements (e.g., UNION, EXCEPT)
)

distinct = None | Distinct

order_by_clause = OrderByClause(expr* expr_list, order order)

order = ASC | DESC

expr = AndExpr(expr* expr_list)
     | OrExpr(expr* expr_list)
     | NotExpr(expr expr)
     | CompareExpr(compare_op op, expr left_value, expr right_value)
     | AggregateExpr(aggregate_op op, expr value, distinct distinct)
     | BinaryExpr(binary_op op, expr left_value, expr right_value)
     | BetweenExpr(expr field, expr left_value, expr right_value)
     | InExpr(column_ref left_value, expr right_value)
     | LikeExpr(column_ref left_value, expr right_value)
     | AllRows(table_ref table_name)
     | select_stmt
     | Literal(string value)
     | ColumnReference(column_ref column_name)

aggregate_op = Sum | Max | Min | Count | Avg

compare_op = LessThan | LessThanEqual | GreaterThan
           | GreaterThanEqual | Equal | NotEqual

binary_op = Add | Sub | Divide | Multiply

compound_stmt = CompoundStatement(compound_op op, select_stmt query)

compound_op = Union | Intersect | Except
```

Figure 2: ASDL Grammar of SemQL used in TranX

another Adam optimizer with a constant learning rate of $1e-3$ for all remaining parameters. During training, we update model parameters for 25000 iterations, and freeze the TABERT parameters at the first 1000 update steps. We use a batch size of 30 and beam size of 3. We use gradient accumulation for large models to fit a batch into GPU memory.

B.2 Weakly-supervised Parsing on WIKITABLEQUESTIONS

Model We use MAPO (Liang et al., 2018), a strong weakly-supervised semantic parser. The original MAPO models comes with an LSTM encoder, which generates utterance and column representations used by the decoder to predict table queries. We directly substitute the encoder with TABERT, and project the utterance and table representations from TABERT to the original embedding space using a linear transformation. MAPO uses a domain-specific query language tailored to answer compositional questions on a single table. For instance, the example question in Fig. 1 could be answered using the following query:

```
Table.contains(column=Position, value=1st)
# Get rows whose 'Position' field contains '1st'
    .argmax(order_by=Year)
# Get the row which has the largest 'Year' field
    .hop(column=Venue)
# Select the value of 'Venue' in the result row
```

MAPO is written in Tensorflow. In our experiments we use an optimized re-implementation in PyTorch, which yields 4× training speedup.

Configuration We use the same optimizer and learning rate schedule as in § B.1. We use a batch size of 10, and train the model for 20000 steps, with the TABERT parameters frozen at the first 5000 steps. Other hyper-parameters are kept the same as the original MAPO system.