
TaBERT: Learning Contextual Representations for Natural Language Utterances and Structured Tables

This repository contains source code for the [TaBERT](#) model, a pre-trained language model for learning joint representations of natural language utterances and (semi-)structured tables for semantic parsing. [TaBERT](#) is pre-trained on a massive corpus of 26M Web tables and their associated natural language context, and could be used as a drop-in replacement of a semantic parsers original encoder to compute representations for utterances and table schemas (columns).

Installation

First, install the conda environment [tabert](#) with supporting libraries.

```
1 bash scripts/setup_env.sh
```

Once the conda environment is created, install [TaBERT](#) using the following command:

```
1 conda activate tabert
2 pip install --editable .
```

Integration with HuggingFace's [pytorch-transformers Library](#) is still WIP. While all the pre-trained models were developed based on the old version of the library [pytorch-pretrained-bert](#), they are compatible with the the latest version [transformers](#). The conda environment will install both versions of the transformers library, and [TaBERT](#) will use [pytorch-pretrained-bert](#) by default. You could uninstall the [pytorch-pretrained-bert](#) library if you prefer using [TaBERT](#) with the latest version of [transformers](#).

Pre-trained Models

Pre-trained models could be downloaded from this Google Drive shared folder. Please uncompress the tarball files before usage.

Pre-trained models could be downloaded from command line as follows: ““shell script pip install gdown

TaBERT_Base_(k=1)

gdown 'https://drive.google.com/uc?id=1-pdtksj9RzC4yEqdrJQaZu4-dlEXZbM9'

TaBERT_Base_(K=3)

gdown 'https://drive.google.com/uc?id=1NPxbGhwJF1uU9EC18YFsEZYE-IQR7ZLj'

TaBERT_Large_(k=1)

gdown 'https://drive.google.com/uc?id=1eLJFUWnrJR06QpROYWKXlbSOjRDDZ3yZ'

TaBERT_Large_(K=3)

gdown 'https://drive.google.com/uc?id=17NTNIqxqYexAzaH_TgEfK42-KmjIRC-g'

```
1
2  ## Using a Pre-trained Model
3
4  To load a pre-trained model from a checkpoint file:
5
6  ```python
7  from table_bert import TableBertModel
8
9  model = TableBertModel.from_pretrained(
10      'path/to/pretrained/model/checkpoint.bin',
11  )
```

To produce representations of natural language text and its associated table:

```
1  from table_bert import Table, Column
2
3  table = Table(
4      id='List of countries by GDP (PPP)',
5      header=[
6          Column('Nation', 'text', sample_value='United States'),
7          Column('Gross Domestic Product', 'real', sample_value='
8              21,439,453')
9      ],
10     data=[
11         ['United States', '21,439,453'],
12         ['China', '27,308,857'],
13         ['European Union', '22,774,165'],
14     ]
15 ).tokenize(model.tokenizer)
16
17 # To visualize table in an IPython notebook:
18 # display(table.to_data_frame(), detokenize=True)
```

```
19 context = 'show me countries ranked by GDP'
20
21 # model takes batched, tokenized inputs
22 context_encoding, column_encoding, info_dict = model.encode(
23     contexts=[model.tokenizer.tokenize(context)],
24     tables=[table]
25 )
```

For the returned tuple, `context_encoding` and `column_encoding` are PyTorch tensors representing utterances and table columns, respectively. `info_dict` contains useful meta information (e.g., context/table masks, the original input tensors to BERT) for downstream application.

```
1 context_encoding.shape
2 >>> torch.Size([1, 7, 768])
3
4 column_encoding.shape
5 >>> torch.Size([1, 2, 768])
```

Use Vanilla BERT To initialize a TaBERT model from the parameters of BERT:

```
1 from table_bert import TableBertModel
2
3 model = TableBertModel.from_pretrained('bert-base-uncased')
```

Example Applications

TaBERT could be used as a general-purpose representation learning layer for semantic parsing tasks over database tables. Example applications could be found under the [examples](#) folder.

Extract/Preprocess Table Corpora from CommonCrawl and Wikipedia

Prerequisite

The following libraries are used for data extraction:

- [jnius](#)
- [info.bliki.wiki](#)
- [wikitextparser](#)
- [Beautiful Soup 4](#)
- Java Wikipedia code located at [contrib/wiki_extractor](#)
 - It compiles to a `.jar` file using maven, which is also included in the folder
- [jdk 12+](#)

Installation

First, you need to install Java JDK. Then use the following command to install necessary Python libraries.

```
1 pip install -r preprocess/requirements.txt
2 python -m spacy download en_core_web_sm
```

Training Table Corpora Extraction

CommonCrawl WDC Web Table Corpus 2015 Details of the dataset could be found at [here](#). We used the English relational tables split, which could be downloaded at [here](#).

The script to preprocess the data is at `scripts/preprocess_commoncrawl_tables.sh`. The following command pre-processes a sample of the whole WDC dataset. To preprocess the whole dataset, simply replace the `input_file` with the root folder of the downloaded tar ball files. ““shell script `mkdir -p data/datasets wget http://data.dws.informatik.uni-mannheim.de/webtables/2015-07/sample.gz -P data/datasets gzip -d < data/datasets/sample.gz > data/datasets/commoncrawl.sample.jsonl`

```
python
-m preprocess.common_crawl
-worker_num 12
-input_file data/datasets/commoncrawl.sample.jsonl
-output_file data/preprocessed_data/common_crawl.preprocessed.jsonl
```

```
1
2 ##### Wikipedia Tables
3
4 The script to extract Wiki tables is at `scripts/extract_wiki_tables.sh`
  `. It demonstrates
5 extracting tables from a sampled Wikipedia dump. Again, you may need
  the full Wikipedida dump
6 to perform data extraction.
7
8 ### Notes for Table Extraction
9
10 **Extract Tables from Scraped HTML Pages**
11 Most code in `preprocess.extract_wiki_data` is for extracting
  surrounding
12 natural language sentences around tables. If you are only interested in
13 extracting tables (e.g., from scraped Wiki Web pages), you could just
  use
14 the `extract_table_from_html` function. See the comments for more
  details.
```

```

15
16 ## Training Data Generation
17
18 This section documents how to generate training data for masked
19   language modeling training
20   from extracted and preprocessed tables.
21
22 The scripts to generate training data for our vanilla `TaBERT(K=1)` and
23   vertical attention
24   `TaBERT(k=3)` models are `utils/generate_vanilla_tabert_training_data.
25   py` and
26   `utils/generate_vertical_tabert_training_data.py`. They are heavily
27   optimized for generating
28   data in parallel in a distributed compute environment, but could still
29   be used locally.
30
31 The following script assumes you have concatenated
32   the `.jsonl` files obtained from running the data extraction scripts on
33   Wikipedia and CommonCrawl
34   corpora and saved to `data/preprocessed_data/tables.jsonl`
35
36 ```shell script
37 cd data/preprocessed_data
38 cat common_crawl.preprocessed.jsonl wiki_tables.jsonl > tables.jsonl

```

The following script generates training data for a vanilla TaBERT(K=1) model: “shell script output_dir=data/train_data/vanilla_tabert mkdir -p \${output_dir}

```

python -m utils.generate_vanilla_tabert_training_data
-output_dir ${output_dir}
-train_corpus data/preprocessed_data/tables.jsonl
-base_model_name bert-base-uncased
-do_lower_case
-epochs_to_generate 15
-max_context_len 128
-table_mask_strategy column
-context_sample_strategy concatenate_and_enumerate
-masked_column_prob 0.2
-masked_context_prob 0.15
-max_predictions_per_seq 200
-cell_input_template 'column|type|value'
-column_delimiter "[SEP]"

```

```

1
2 The following script generates training data for a `TaBERT(K=3)` model
   with

```

```
3 vertical self-attention:
4 ```shell script
5 output_dir=data/train_data/vertical_tabert
6 mkdir -p ${output_dir}
7
8 python -m utils.generate_vertical_tabert_training_data \
9     --output_dir ${output_dir} \
10    --train_corpus data/preprocessed_data/tables.jsonl \
11    --base_model_name bert-base-uncased \
12    --do_lower_case \
13    --epochs_to_generate 15 \
14    --max_context_len 128 \
15    --table_mask_strategy column \
16    --context_sample_strategy concatenate_and_enumerate \
17    --masked_column_prob 0.2 \
18    --masked_context_prob 0.15 \
19    --max_predictions_per_seq 200 \
20    --cell_input_template 'column|type|value' \
21    --column_delimiter "[SEP]"
```

Parallel Data Generation The script has two additional arguments, `--global_rank` and `--world_size`. To generate training data in parallel using `N` processes, just fire up `N` processes with the same set of arguments and `--world_size=N`. The argument `--global_rank` is set to `[1, 2, ..., N]` for each process.

Model Training

Our models are trained on a cluster of 32GB Tesla V100 GPUs. The following script demonstrates training a vanilla `TabERT` (`k=1`) model using a single GPU with gradient accumulation: ““shell script `mkdir -p data/runs/vanilla_tabert`

```
python train.py
-task vanilla
-data-dir data/train_data/vanilla_tabert
-output-dir data/runs/vanilla_tabert
-table-bert-extra-config '{}'
-train-batch-size 8
-gradient-accumulation-steps 32
-learning-rate 2e-5
-max-epoch 10
-adam-eps 1e-08
-weight-decay 0.0
-fp16
```

`-clip-norm 1.0`
`-empty-cache-freq 128`

```
1
2 The following script shows training a `TaBERT(k=3)` model with vertical
  self-attention:
3 ```shell script
4 mkdir -p data/runs/vertical_tabert
5
6 python train.py \
7     --task vertical_attention \
8     --data-dir data/train_data/vertical_tabert \
9     --output-dir data/runs/vertical_tabert \
10    --table-bert-extra-config '{"base_model_name": "bert-base-uncased",
    "num_vertical_attention_heads": 6, "num_vertical_layers": 3, "
    predict_cell_tokens": true}' \
11    --train-batch-size 8 \
12    --gradient-accumulation-steps 64 \
13    --learning-rate 4e-5 \
14    --max-epoch 10 \
15    --adam-eps 1e-08 \
16    --weight-decay 0.01 \
17    --fp16 \
18    --clip-norm 1.0 \
19    --empty-cache-freq 128
```

Distributed training with multiple GPUs is similar to XLM.

Reference

If you plan to use TaBERT in your project, please consider citing our paper:

```
1 @inproceedings{yin20acl,
2   title = {Ta{BERT}: Pretraining for Joint Understanding of Textual
3     and Tabular Data},
4   author = {Pengcheng Yin and Graham Neubig and Wen-tau Yih and
5     Sebastian Riedel},
6   booktitle = {Annual Conference of the Association for Computational
7     Linguistics (ACL)},
8   month = {July},
9   year = {2020}
10 }
```

License

TaBERT is CC-BY-NC 4.0 licensed as of now.