BERT (Bidirectional Encoder Representations from Transformers)

WARNING: We are on the way to deprecate most of the code in this directory. Please see this link for the new tutorial and use the new code in `nlp/modeling`. This README is still correct for this legacy implementation.

The academic paper which describes BERT in detail and provides full results on a number of tasks can be found here: https://arxiv.org/abs/1810.04805.

This repository contains TensorFlow 2.x implementation for BERT.

Contents

Pre-trained Models

We released both checkpoints and tf.hub modules as the pretrained models for fine-tuning. They are TF 2.x compatible and are converted from the checkpoints released in TF 1.x official BERT repository google-research/bert in order to keep consistent with BERT paper.

Access to Pretrained Checkpoints

Pretrained checkpoints can be found in the following links:

Note: We have switched BERT implementation to use Keras functional-style networks in nlp/modeling. The new checkpoints are:

- **`BERT-Large, Uncased (Whole Word Masking)`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Large, Cased (Whole Word Masking)`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Uncased`**: 12-layer, 768-hidden, 12-heads, 110M parameters
- **`BERT-Large, Uncased`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Cased`**: 12-layer, 768-hidden, 12-heads , 110M parameters
- **`BERT-Large, Cased`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Multilingual Cased`**: 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters

We recommend to host checkpoints on Google Cloud storage buckets when you use Cloud GPU/TPU.

Restoring from Checkpoints

`tf.train.Checkpoint` is used to manage model checkpoints in TF 2. To restore weights from provided pre-trained checkpoints, you can use the following code:

```
init_checkpoint='the pretrained model checkpoint path.'
model=tf.keras.Model() # Bert pre-trained model as feature extractor.
checkpoint = tf.train.Checkpoint(model=model)
checkpoint.restore(init_checkpoint)
```

Checkpoints featuring native serialized Keras models (i.e. model.load()/load_weights()) will be available soon.

Access to Pretrained hub modules.

Pretrained tf.hub modules in TF 2.x SavedModel format can be found in the following links:

- **`BERT-Large, Uncased (Whole Word Masking)`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Large, Cased (Whole Word Masking)`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Uncased`**: 12-layer, 768-hidden, 12-heads, 110M parameters
- **`BERT-Large, Uncased`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Cased`**: 12-layer, 768-hidden, 12-heads , 110M parameters
- **`BERT-Large, Cased`**: 24-layer, 1024-hidden, 16-heads, 340M parameters
- **`BERT-Base, Multilingual Cased`**: 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- **`BERT-Base, Chinese`**: Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

Set Up

```
export PYTHONPATH="$PYTHONPATH:/path/to/models"
```

Install `tf-nightly` to get latest updates:

```
pip install tf-nightly-gpu
```

With TPU, GPU support is not necessary. First, you need to create a `tf-nightly` TPU with ctpu tool:

```
ctpu up -name <instance name> --tf-version="nightly"
```

Second, you need to install TF 2 `tf-nightly` on your VM:

```
pip install tf-nightly
```

Process Datasets

Pre-training

There is no change to generate pre-training data. Please use the script `../data/create_pretraining_data.py` which is essentially branched from BERT research repo to get processed pre-training data and it adapts to TF2 symbols and python3 compatibility.

Running the pre-training script requires an input and output directory, as well as a vocab file. Note that max_seq_length will need to match the sequence length parameter you specify when you run pre-training.

Example shell script to call create_pretraining_data.py

```
export WORKING_DIR='local disk or cloud location'
export BERT_DIR='local disk or cloud location'
python models/official/nlp/data/create_pretraining_data.py \
  --input_file=$WORKING_DIR/input/input.txt \
  --output_file=$WORKING_DIR/output/tf_examples.tfrecord \
  --vocab_file=$BERT_DIR/wwm_uncased_L-24_H-1024_A-16/vocab.txt \
  --do_lower_case=True \
  --max_seq_length=512 \
  --max_predictions_per_seq=76 \
  --masked_lm_prob=0.15 \
  --random_seed=12345 \
  --dupe_factor=5
```

Fine-tuning

To prepare the fine-tuning data for final model training, use the `../data/create_finetuning_data.py` script. Resulting datasets in `tf_record` format and training meta data should be later passed to training or evaluation scripts. The task-specific arguments are described in following sections:

- GLUE

Users can download the GLUE data by running this script and unpack it to some directory `$GLUE_DIR`. Also, users can download Pretrained Checkpoint and locate on some directory `$BERT_DIR` instead of using checkpoints on Google Cloud Storage.

```
export GLUE_DIR=~/glue
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16

export TASK_NAME=MNLI
export OUTPUT_DIR=gs://some_bucket/datasets
python ../data/create_finetuning_data.py \
 --input_data_dir=${GLUE_DIR}/${TASK_NAME}/ \
 --vocab_file=${BERT_DIR}/vocab.txt \
```

```
 --train_data_output_path=${OUTPUT_DIR}/${TASK_NAME}_train.tf_record \
 --eval_data_output_path=${OUTPUT_DIR}/${TASK_NAME}_eval.tf_record \
 --meta_data_file_path=${OUTPUT_DIR}/${TASK_NAME}_meta_data \
 --fine_tuning_task_type=classification --max_seq_length=128 \
 --classification_task_name=${TASK_NAME}
```

- SQUAD

The SQuAD website contains detailed information about the SQuAD datasets and evaluation.

The necessary files can be found here:

- train-v1.1.json
- dev-v1.1.json
- evaluate-v1.1.py
- train-v2.0.json
- dev-v2.0.json
- evaluate-v2.0.py

```
export SQUAD_DIR=~/squad
export SQUAD_VERSION=v1.1
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export OUTPUT_DIR=gs://some_bucket/datasets

python ../data/create_finetuning_data.py \
 --squad_data_file=${SQUAD_DIR}/train-${SQUAD_VERSION}.json \
 --vocab_file=${BERT_DIR}/vocab.txt \
 --train_data_output_path=${OUTPUT_DIR}/squad_${SQUAD_VERSION}_train.tf_record \
 --meta_data_file_path=${OUTPUT_DIR}/squad_${SQUAD_VERSION}_meta_data \
 --fine_tuning_task_type=squad --max_seq_length=384
```

Note: To create fine-tuning data with SQUAD 2.0, you need to add flag `--version_2_with_negative=True`.

Fine-tuning with BERT

Cloud GPUs and TPUs

- Cloud Storage

The unzipped pre-trained model files can also be found in the Google Cloud Storage folder `gs://cloud-tpu-checkpoints/bert/keras_bert`. For example:

```
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export MODEL_DIR=gs://some_bucket/my_output_dir
```

Currently, users are able to access to `tf-nightly` TPUs and the following TPU script should run with `tf-nightly`.

- GPU -> TPU

Just add the following flags to `run_classifier.py` or `run_squad.py`:

```
--distribution_strategy=tpu
--tpu=grpc://${TPU_IP_ADDRESS}:8470
```

Sentence and Sentence-pair Classification Tasks

This example code fine-tunes `BERT-Large` on the Microsoft Research Paraphrase Corpus (MRPC) corpus, which only contains 3,600 examples and can fine-tune in a few minutes on most GPUs.

We use the `BERT-Large` (uncased_L-24_H-1024_A-16) as an example throughout the workflow. For GPU memory of 16GB or smaller, you may try to use `BERT-Base` (uncased_L-12_H-768_A-12).

```
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export MODEL_DIR=gs://some_bucket/my_output_dir
export GLUE_DIR=gs://some_bucket/datasets
export TASK=MRPC

python run_classifier.py \
  --mode='train_and_eval' \
  --input_meta_data_path=${GLUE_DIR}/${TASK}_meta_data \
  --train_data_path=${GLUE_DIR}/${TASK}_train.tf_record \
  --eval_data_path=${GLUE_DIR}/${TASK}_eval.tf_record \
  --bert_config_file=${BERT_DIR}/bert_config.json \
  --init_checkpoint=${BERT_DIR}/bert_model.ckpt \
  --train_batch_size=4 \
  --eval_batch_size=4 \
  --steps_per_loop=1 \
  --learning_rate=2e-5 \
  --num_train_epochs=3 \
  --model_dir=${MODEL_DIR} \
  --distribution_strategy=mirrored
```

Alternatively, instead of specifying `init_checkpoint`, you can specify `hub_module_url` to employ a pretraind BERT hub module, e.g., `--hub_module_url=https://tfhub.dev/tensorflow/bert_en_uncased_L-24_H-1024_A-16/1`.

After training a model, to get predictions from the classifier, you can set the `--mode=predict` and offer the test set tfrecords to `--eval_data_path`. Output will be created in file called test_results.tsv in the output folder. Each line will contain output for each sample, columns are the class probabilities.

```
python run_classifier.py \
  --mode='predict' \
  --input_meta_data_path=${GLUE_DIR}/${TASK}_meta_data \
  --eval_data_path=${GLUE_DIR}/${TASK}_eval.tf_record \
  --bert_config_file=${BERT_DIR}/bert_config.json \
```

```
  --eval_batch_size=4 \
  --model_dir=${MODEL_DIR} \
  --distribution_strategy=mirrored
```

To use TPU, you only need to switch distribution strategy type to `tpu` with TPU information and use remote storage for model checkpoints.

```
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export TPU_IP_ADDRESS='???'
export MODEL_DIR=gs://some_bucket/my_output_dir
export GLUE_DIR=gs://some_bucket/datasets
export TASK=MRPC

python run_classifier.py \
  --mode='train_and_eval' \
  --input_meta_data_path=${GLUE_DIR}/${TASK}_meta_data \
  --train_data_path=${GLUE_DIR}/${TASK}_train.tf_record \
  --eval_data_path=${GLUE_DIR}/${TASK}_eval.tf_record \
  --bert_config_file=${BERT_DIR}/bert_config.json \
  --init_checkpoint=${BERT_DIR}/bert_model.ckpt \
  --train_batch_size=32 \
  --eval_batch_size=32 \
  --steps_per_loop=1000 \
  --learning_rate=2e-5 \
  --num_train_epochs=3 \
  --model_dir=${MODEL_DIR} \
  --distribution_strategy=tpu \
  --tpu=grpc://${TPU_IP_ADDRESS}:8470
```

Note that, we specify `steps_per_loop=1000` for TPU, because running a loop of training steps inside a `tf.function` can significantly increase TPU utilization and callbacks will not be called inside the loop.

SQuAD 1.1

The Stanford Question Answering Dataset (SQuAD) is a popular question answering benchmark dataset. See more in SQuAD website.

We use the `BERT-Large` (uncased_L-24_H-1024_A-16) as an example throughout the workflow. For GPU memory of 16GB or smaller, you may try to use `BERT-Base` (uncased_L-12_H-768_A-12).

```
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export SQUAD_DIR=gs://some_bucket/datasets
export MODEL_DIR=gs://some_bucket/my_output_dir
export SQUAD_VERSION=v1.1

python run_squad.py \
```

```
  --input_meta_data_path=${SQUAD_DIR}/squad_${SQUAD_VERSION}_meta_data \
  --train_data_path=${SQUAD_DIR}/squad_${SQUAD_VERSION}_train.tf_record \
  --predict_file=${SQUAD_DIR}/dev-v1.1.json \
  --vocab_file=${BERT_DIR}/vocab.txt \
  --bert_config_file=${BERT_DIR}/bert_config.json \
  --init_checkpoint=${BERT_DIR}/bert_model.ckpt \
  --train_batch_size=4 \
  --predict_batch_size=4 \
  --learning_rate=8e-5 \
  --num_train_epochs=2 \
  --model_dir=${MODEL_DIR} \
  --distribution_strategy=mirrored
```

Similarily, you can replace `init_checkpoint` FLAG with `hub_module_url` to specify a hub module path.

`run_squad.py` writes the prediction for `--predict_file` by default. If you set the `--model=predict` and offer the SQuAD test data, the scripts will generate the prediction json file.

To use TPU, you need switch distribution strategy type to `tpu` with TPU information.

```
export BERT_DIR=gs://cloud-tpu-checkpoints/bert/keras_bert/uncased_L-24_H-1024_A-16
export TPU_IP_ADDRESS='???'
export MODEL_DIR=gs://some_bucket/my_output_dir
export SQUAD_DIR=gs://some_bucket/datasets
export SQUAD_VERSION=v1.1

python run_squad.py \
  --input_meta_data_path=${SQUAD_DIR}/squad_${SQUAD_VERSION}_meta_data \
  --train_data_path=${SQUAD_DIR}/squad_${SQUAD_VERSION}_train.tf_record \
  --predict_file=${SQUAD_DIR}/dev-v1.1.json \
  --vocab_file=${BERT_DIR}/vocab.txt \
  --bert_config_file=${BERT_DIR}/bert_config.json \
  --init_checkpoint=${BERT_DIR}/bert_model.ckpt \
  --train_batch_size=32 \
  --learning_rate=8e-5 \
  --num_train_epochs=2 \
  --model_dir=${MODEL_DIR} \
  --distribution_strategy=tpu \
  --tpu=grpc://${TPU_IP_ADDRESS}:8470
```

The dev set predictions will be saved into a file called predictions.json in the model_dir:

```
python $SQUAD_DIR/evaluate-v1.1.py $SQUAD_DIR/dev-v1.1.json ./squad/predictions.json
```