BERT QA on Electronics Medical Record dataset

This folder contains python scripts to train the BERT's language model using clinical notes (for domain adaptation), further create tfrecords for emrQA dataset and use tfrecords to finetune BERT QA model and save checkpoints for every epoch, and then use another script to plot loss function and evaluation metrics for saved checkpoints.

Note : Always used max_sequence length as 512 (as clinical notes are longer)

Below is some information of usage of each python script and its purpose :

1. The process of adapting BERT for LM training can be divided into following stages :
    - Create tfrecords for LM training using only clinical notes from emrQA dataset (can either use BERT's original vocab or replace the placeholder's in BERT's vocab with most frequent 995 words in clinical notes) - CPU/GPU
    - Train the BERT's LM using tfrecords from above step on GPU
    - Create tfrecords for finetuning from emrQA dataset (clinical notes, questions and answers) - CPU/GPU
    - Finetune BERT's QA model using tfrecords generated from previous step (optional step before this step could be first finetune BERT-QA model on SQUAD dataset). Checkpoints after every epoch are saved during finetuning - TPU
    - Evaluate the checkpoints saved during finetuning to get the loss function plot on tensorboard and see the evaluation metrics on console (not on tensorboard)

    For LM training, use pre-training-train.txt and pre-training-val.txt to generate tfrecords. For finetuning, use emrqa_train_data.json and emrqa_val_data.json
2. Install the following pip dependencies for BERT (on top of a GPU enabled TensorFlow install):

    ```
    pip install bert-tensorflow
    ```

3. If using TPU, create a bert_model folder in google cloud bucket else create a folder on local machine. It should contain BERT's config file, vocab file (can be BERT's original one or the modified one with placeholder being replaced with domain specific words) and a tensorFlow checkpoint (`bert_model.ckpt`) containing the pre-trained weights (which is actually 3 files).

Below code snippet can be used to authenticate and connect to TPU and get the TPU's ip address :

```
import datetime
import json
import os
import pprint
import random
import string
import sys
import tensorflow as tf

assert 'COLAB_TPU_ADDR' in os.environ, 'ERROR: Not connected to a TPU
runtime; please see the first cell in this notebook for instructions!'
TPU_ADDRESS = 'grpc://' + os.environ['COLAB_TPU_ADDR']
print('TPU address is', TPU_ADDRESS)

from google.colab import auth
auth.authenticate_user()
with tf.Session(TPU_ADDRESS) as session:
  print('TPU devices:')
  pprint.pprint(session.list_devices())

  # Upload credentials to TPU
  with open('/content/adc.json', 'r') as f:
    auth_info = json.load(f)
  tf.contrib.cloud.configure_gcs(session, credentials=auth_info)
  # Now credentials are set for all future sessions on this TPU.
```

4. Create another folder emr-data which contains two text files (one for training LM and another for evaluating LM). The text file should have one sentence from clinical note per line. Every clinical note is delimited by empty lines.

5. In emr-data folder, add json files from emrQA dataset for training (finetuning) and evaluation. Json files contain clinical notes, ids and question-answer pairs.

6. In order to create tfrecords for LM training, run following command :

```
python prepare_LMtraining_emr_tfrecords.py \
  --input_file=$EMR-DATA-DIR-PATH/emr_lmtrain_train.txt \
  --output_file=/emr_lmtrain_training.tfrecord \
  --vocab_file=$BERT_BASE_DIR/vocab.txt \
  --do_lower_case=True \
  --max_seq_length=512 \
  --max_predictions_per_seq=77 \
  --masked_lm_prob=0.15 \
  --random_seed=12345 \
```

```
  --dupe_factor=5
```

Similarly, run above command to generate tfrecords for validation data text file.

7. Now, we have tfrecords for training and evaluating LM. We only use word masking task to train LM as NSP task is not appropriate for clinical notes. To train and evaluate the LM (after every epoch) on GPU, run following command :

```
python pretrain_bert_on_emr.py \
    --input_file_train=~/emr-data/tf_train.tfrecord \
    --input_file_eval=~/emr-data/tf_test.tfrecord \
    --output_dir=~/pretrain_output/ \
    --do_train=True \
    --do_eval=True \
    --bert_config_file=~/bert_model/bert_config.json \
    --init_checkpoint=~/bert_model/bert_model.ckpt \
    --vocab_file=~/bert_model/vocab.txt \
    --num_train_epochs=5 \
    --train_batch_size=8 \
    --eval_batch_size=8 \
    --max_seq_length=512 \
    --max_predictions_per_seq=77 \
    --learning_rate=2e-5 \
    --warmup_proportion=0.1 \
    --save_ckpt_every_n_epochs=1 \
    --iterations_per_loop=100 \
    --eval_on_train=True
```

The pros of running above command is that it saves checkpoint, evaluates it after every epoch and generates tensorboard summary. But pretraining on GPU is pretty slow and the batch size cannot be increased beyond 8 for GPU with 16 GB. Below script can be used to do pretraining on TPU (but this script might have some small bugs as I did not take the backup for this one, so re-wrote it and haven't tested)

```
python pretrain_bert_emr_tpu.py \
    --input_file=$EMR_DATA_BUCKET/tf_train.tfrecord \
    --eval_input_file=$EMR_DATA_BUCKET/tf_test.tfrecord \
    --output_dir=$OUTPUT_BUCKET/ \
    --do_train=True \
    --do_eval=True \
    --bert_config_file=$BERT_MODEL_BUCKET/bert_config.json \
    --init_checkpoint=$BERT_MODEL_BUCKET/bert_model.ckpt \
    --vocab_file=$BERT_MODEL_BUCKET/vocab.txt \
    --num_train_epochs=5 \
    --train_batch_size=64 \
    --eval_batch_size=8 \
```

```
    --max_seq_length=512 \
    --max_predictions_per_seq=77 \
    --learning_rate=2e-5 \
    --warmup_proportion=0.1 \
    --use_tpu=True \
    --tpu_name=grpc://10.47.188.226:8470 \
```

8. Generate tfrecords of QA dataset which can be used to finetune the BERT-QA model. Following command can be used for the same (on CPU or GPU) :

```
python prepare_finetuning_emrQA_tfrecords.py \
     --bert_config_file=$BERT_MODEL_FOLDER/bert_config.json \
     --vocab_file=$BERT_MODEL_FOLDER/vocab.txt \
     --max_seq_length=512 \
     --input_json=emr_data_train.json \
     --output_tfrecord=emr-train.tfrecords \
     --is_training=True \
```

   Similarly, tfrecords for prediction and evaluation can be generated by setting is_training as False.

9. Checkpoints of BERT with language model trained on domain-specific corpus can now be used to initialize the weights of BERT-QA model. The BERT can be used for QA with long context in two ways : 1) by predicting the answer for every sub-part (split) of clinical note and then picking the predicted answer span with maximum start_logit + end_logit. 2) by shortlisting the candidate splits of given clinical note which may contain answer. This is done by a classifier which given a question and sub-part (split) from clinical note would predict if this sub-part might contain answer or not.

   For 1), the following command can be used to train the model and save checkpoints after every epoch (on TPU) :

```
python run_emr_tpu \
  --logtostderr \
  --bert_config_file=$BERT_MODEL_BUCKET/bert_config.json \
  --vocab_file=$BERT_MODEL_BUCKET/vocab.txt
  --train_precomputed_file=$EMR_DATA_BUCKET_BUCKET/emr-train.tfrecords \
  --train_num_precomputed=466703 \
  --save_checkpoints_steps=7292 \
  --iterations_per_loop=7292 \
  --learning_rate=3e-5 \
  --num_train_epochs=4 \
  --max_seq_length=512 \
  --train_batch_size=64 \
  --init_checkpoint=$LMTraining_OUTPUT_BUCKET/bert_model.ckpt \
  --do_train \
```

```
  --output_dir=$FINETUNING_OUTPUT_BUCKET/ \
  --use_tpu=True \
  --tpu_name=grpc://10.47.188.226:8470 \
```

Tpu_name is the ip address of TPU, init_checkpoint is the path to checkpoint of BERT after LM training, train_num_precomputed is number of examples in tfrecords of training data (prepare_finetuning_emrQA_tfrecords.py script prints after generating tfrecords), save_checkpoints_steps and iterations_per_loop should be assigned as number of steps in one epoch and can be computed by int(train_num_precomputed/train_batch_size)

Once finetuned with QA dataset and saved checkpoints for every epoch, below command can be run to evaluate the checkpoints and get plots for loss function :

```
python predict_and_evaluate \
  --bert_config_file=$BERT_MODEL_BUCKET/bert_config.json \
  --vocab_file=$BERT_MODEL_BUCKET/vocab.txt \
  --train_precomputed_file=$EMR_DATA_BUCKET/emr-train.tfrecords \
  --train_num_precomputed=466703 \
  --predict_precomputed_file=$EMR_DATA_BUCKET/eval.tf_record \
  --eval_num_precomputed=45563 \
  --eval_precomputed_file=$EMR_DATA_BUCKET/eval.tf_record \
  --learning_rate=3e-5 \
  --num_train_epochs=7 \
  --max_seq_length=512 \
  --save_checkpoints_steps=5000 \
  --init_checkpoint=$FINETUNING_OUTPUT_BUCKET/model.ckpt-7264 \
  --do_eval \
  --predict_file=/content/language/emr_test_data.json \
  --output_prediction_file=$FINETUNING_OUTPUT_BUCKET/predictions.json \
--evaluation_metrics_file=$FINETUNING_OUTPUT_BUCKET/evaluation_metrics.js
on
  --output_dir=g$FINETUNING_OUTPUT_BUCKET/ \
  --use_tpu=True \
  --tpu_name=grpc://10.71.4.18:8470 \
```

For 2), above two commands can be used and python filename needs to be replaced by run_emr_tpu_with_classifier.py and predict_and_evaluate_with_classifier.py