HPC-AI Competition BERT-LARGE Benchmark Guidelines

1 AI Part: GLUE benchmark fine-tuning with Tensorflow BERT-Large

1.1 About the application and benchmarks

1.1.1 About BERT-Large

BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

BERT is a method of pre-training language representations, meaning that we train a general-purpose "language understanding" model on a large text corpus (like Wikipedia), and then use that model for downstream NLP tasks that we care about (like question answering). BERT outperforms previous methods because it is the first unsupervised, deeply bidirectional system for pre-training NLP.

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017) and released in the tensor2tensor library. The architecture of BERT is almost identical to the original Transformer. A good reference guides for its implementation is "The Annotated Transformer."

The developer team denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. The team primarily report results on two model sizes:

- BERT-BASE (L=12, H=768, A=12, Total Parameters=110M) and
- BERT-LARGE (L=24, H=1024, A=16, Total Parameters=340M).

BERT-BASE contains 110M parameters and BERT-LARGE contains 340M parameters.

1.1.2 About Benchmark Datasets GLUE

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems.

The format of the GLUE benchmark is model-agnostic, so any system capable of processing sentence and sentence pairs and producing corresponding predictions is eligible to participate. The benchmark tasks are selected so as to favor models that share information across tasks using parameter sharing or other transfer learning techniques. The ultimate goal of GLUE is to drive research in the development of general and robust natural language understanding systems.

1.1.3 About GLUE/MNLI

Config description: The Multi-Genre Natural Language Inference Corpus is a crowdsourced collection of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral). The premise sentences are gathered from ten different sources, including transcribed speech, fiction, and government reports. We use the standard test set, for which we obtained private labels from the authors, and evaluate on both the matched (in-domain) and mismatched (cross-domain) section. We also use and recommend the SNLI corpus as 550k examples of auxiliary training data.

Homepage: http://www.nyu.edu/projects/bowman/multinli/

Dataset Download size: 298.29 MiB

1.2 Running GLUE benchmarks

1.2.1 Download the benchmark

1.2.1.1 Download the dataset

Cases in GLUE benchmark are Sentence (and sentence-pair) classification tasks.

Before running the benchmark you must download the GLUE data by running https://gist.github.com/W4ngatang/60c2bdb54d156a41194446737ce03e2e and unpack it to some directory $GLUE_DIR. Next, download the BERT-Large checkpoint and unzip it to some directory $BERT_LARGE_DIR.

If you would like to download all of GLUE datasets, replace "MNLI" with "ALL"

```
[~]# wget
https://gist.githubusercontent.com/W4ngatang/60c2bdb54d156a41194446737ce03e2e/raw/17b8dd0d724281ed7c3b2aeeda662b92809aadd5/download_glue_data.py
[~]# python3 download_glue_data.py --data_dir glue_data --tasks MNLI
```

1.2.1.2 Download the benchmark codes

NVIDIA BERT codes is a publicly available implementation of BERT. It supports Multi-GPU training with Horovod - NVIDIA BERT fine-tune code uses Horovod to implement efficient multi-GPU training with NCCL.

```
[~]# git clone https://github.com/NVIDIA/DeepLearningExamples.git
```

1.2.1.3 Download BERT-Large model file

The BERT-Large, Uncased (Whole Word Masking) model file contains 24-layer, 1024-hidden, 16-heads, 340M parameters. Its download link can be found at https://github.com/google-research/bert

```
[~]# wget
https://storage.googleapis.com/bert_models/2019_05_30/wwm_uncased_L-24_H-1024_A-16.zip
```

1.2.2 How to configure the benchmark

1.2.2.1 Install CUDA developer kit and NCCL libraries

Install NVIDIA Machine Learning repo and get nccl libs, for building NCCL Horovod.

```
[~]# yum install -y https://developer.download.nvidia.com/compute/machine-learning/repos/rhel7/x86_64/nvidia-machine-learning-repo-rhel7-1.0.0-1.x86_64.rpm \
https://developer.download.nvidia.cn/compute/cuda/repos/rhel7/x86_64/cuda-rhel7.repo
[~]# yum install -y libnccl*2.5.6-1+cuda10.0
```

1.2.2.2 Install TensorFlow-GPU

Install prebuilt or customize-built TensorFlow GPU package to run BERT TensorFlow codes.

```
[~]# pip3 install tensorflow-gpu==1.15
```

1.2.2.3 Install MOFED driver

Install MOFED driver for enabling RDMA connection.

```
[~]# ./mlnxofedinstall --enable-mlnx_tune --enable-affinity --all -vvv
```

1.2.2.4 Install NCCL Horovod

Install Horovod to add RDMA and multi-node ability to TensorFlow-GPU.

```
[~]# module load /applications/hpcx-v2.6.0-gcc-MLNX_OFED_LINUX-5.0-1.0.0.0-redhat7.7-x86_64/modulefiles/hpcx-mt-ompi
[~]# HOROVOD_GPU_ALLREDUCE=NCCL \
     HOROVOD_NCCL_HOME=/usr/lib/x86_64-linux-gnu \
     pip3 install --no-cache-dir horovod==0.16
```

So far, you will have

- TensorFlow-GPU installed, with support of NCCL Horovod
- The BERT Large model file
- The NMLI dataset for fine-tune training tasks
- MLNX OFED installed
- A copy of workable OpenMPI-4 at /usr/mpi/gcc/openmpi-4.0.2rc3/bin/mpirun, installed by OFED driver

1.2.3 How to run the benchmark

The following example code fine-tunes BERT-Large on the Multi-Genre Natural Language Inference (MNLI) Corpus, which only contains 392,702 examples and can fine-tune in a few hours on most GPUs.

1.2.3.1 Define the workspace variables

Export the environment variables to specify the PATHs of the dataset, model, codes and results

```
[~]# MPIBINPATH=/usr/mpi/gcc/openmpi-4.0.2rc3/bin \
BERT_WORKSPACE=~/bert \
```

```
RESULT_PATH=$BERT_WORKSPACE/results  \
MNLI_PATH=$BERT_WORKSPACE/data/glue_data/MNLI \
BERT_BASE_MODEL_PATH=$BERT_WORKSPACE/model/uncased_L-12_H-768_A-12 \
BERT_LARGE_MODEL_PATH=$BERT_WORKSPACE/model/wwm_uncased_L-24_H-1024_A-16 \
GOOGLE_BERT_CODE_PATH=$BERT_WORKSPACE/github/bert \
NVIDIA_BERT_CODE_PATH=$BERT_WORKSPACE/github/DeepLearningExamples/TensorFlow
/LanguageModeling/BERT \
TASK_NAME=MNLI \
DATA_PATH=$MNLI_PATH \
MODEL_PATH=$BERT_LARGE_MODEL_PATH \
CODE_PATH=$NVIDIA_BERT_CODE_PATH
```

1.2.3.2 Prepare the files

Download dataset files to $MNLI_PATH

```
[~]# python3 download_glue_data.py --data_dir glue_data --tasks MNLI --
data_dir $MNLI_PATH
```

Unzip model files to $BERT_LARGE_MODEL_PATH

```
[~]# unzip wwm_uncased_L-24_H-1024_A-16.zip -d $BERT_WORKSPACE/model
```

1.2.3.3 Run the benchmark

```
[~]# module load /applications/hpcx-v2.6.0-gcc-MLNX_OFED_LINUX-5.0-1.0.0.0-
redhat7.7-x86_64/modulefiles/hpcx-mt-ompi
[~]# time -p $MPIBINPATH/mpirun -np 4 -H ops003:2,ops004:2 \
-mca btl_tcp_if_include enp3s0f1 -npernode 2 \
-bind-to none -map-by slot -mca pml ob1 -mca btl ^openib \
-x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH \
python $CODE_PATH/run_classifier.py  \
--do_train=true --do_eval=false --do_predict=false \
--task_name=$TASK_NAME --data_dir=$DATA_PATH \
--vocab_file=$MODEL_PATH/vocab.txt \
--bert_config_file=$MODEL_PATH/bert_config.json \
--init_checkpoint=$MODEL_PATH/bert_model.ckpt \
--output_dir=$RESULT_PATH \
--learning_rate=5e-5 --num_train_epochs=0.001 \
--max_seq_length=128 --train_batch_size=32 \
--num_accumulation_steps=1 --save_checkpoints_steps=1000 \
--warmup_proportion=0.1 --use_fp16 --horovod \
2>&1 | tee py36traininglog
```

This very short benchmark job on 0.1% of MNLI dataset will finish in 182 seconds and produce an output like this:

```
INFO:tensorflow:Saving checkpoints for 3 into
/global/home/users/pengzhiz/ops/bert/results/model.ckpt.
I0318 02:32:55.269347 47103642754688 basic_session_run_hooks.py:606] Saving
checkpoints for 3 into
/global/home/users/pengzhiz/ops/bert/results/model.ckpt.
I0318 02:32:58.778598 47103642754688 estimator.py:371] Loss for final step:
1.1915715.
INFO:tensorflow:Total Training Time = 72.98 for Sentences = 384
I0318 02:32:58.779872 47103642754688 run_classifier.py:566] Total Training
Time = 72.98 for Sentences = 384
INFO:tensorflow:Throughput Average (sentences/sec) = 673.53
I0318 02:32:58.780167 47103642754688 run_classifier.py:570] Throughput
Average (sentences/sec) = 673.53
-------------
real 182.25
user 424.65
sys 59.26
```

1.2.3.4 How to evaluate the benchmark

```
[~]# time -p python $CODE_PATH/run_classifier.py \
--do_train=false --do_eval=true --do_predict=false \
--task_name=$TASK_NAME \
--data_dir=$DATA_PATH \
--vocab_file=$MODEL_PATH/vocab.txt \
--bert_config_file=$MODEL_PATH/bert_config.json \
--output_dir=$RESULT_PATH \
--use_fp16
```

The evaluation job will produce an output like this:

```
***** Eval results *****
  eval_accuracy = 0.845588
  eval_loss = 0.505248
  global_step = 343
  loss = 0.505248
```

In the results, accuracy is the higher the better while loss is the lower the better.