Model Garden NLP Common Training Driver

train.py is the common training driver that supports multiple NLP tasks (e.g., pre-training, GLUE and SQuAD fine-tuning etc) and multiple models (e.g., BERT, ALBERT, MobileBERT etc).

Experiment Configuration

train.py is driven by configs defined by the ExperimentConfig including configurations for `task`, `trainer` and `runtime`. The pre-defined NLP related ExperimentConfig can be found in configs/experiment_configs.py.

Experiment Registry

We use an experiment registry to build a mapping between experiment type to experiment configuration instance. For example, configs/finetuning_experiments.py registers `bert/sentence_prediction` and `bert/squad` experiments. User can use `--experiment` FLAG to invoke a registered experiment configuration, e.g., `--experiment=bert/sentence_prediction`.

Overriding Configuration via Yaml and FLAGS

The registered experiment configuration can be overridden by one or multiple Yaml files provided by `--config_file` FLAG. For example:

```
--config_file=configs/experiments/glue_mnli_matched.yaml \
--config_file=configs/models/bert_en_uncased_base.yaml
```

In addition, experiment configuration can be further overriden by `params_override` FLAG. For example:

```
 --params_override=task.train_data.input_path=/some/path,task.hub_module_url=/some/tfhub
```

Run locally on GPUs

An example command for training a model on local GPUs is below. This command trains a BERT-base model on GLUE/MNLI-matched which is a sentence prediction task.

```
PARAMS=runtime.distribution_strategy=mirrored  # Train no GPU
PARAMS=${PARAMS},task.train_data.input_path=/path-to-your-training-data/

python3 train.py \
  --experiment=bert/sentence_prediction \
  --mode=train \
  --model_dir=/a-folder-to-hold-checkpoints-and-logs/ \
  --config_file=configs/models/bert_en_uncased_base.yaml \
```

```
  --config_file=configs/experiments/glue_mnli_matched.yaml \
  --params_override=${PARAMS}
```

Note that you can specify any detailed configuration by appending to the `PARAMS` variable. For example, if you want to load from a pretrained checkpoint as initialization (instead of random initialization):

```
PARAMS=${PARAMS},task.hub_module_url=https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768
```

The configuration entry `task.hub_module_url` uses a URL to a TF-Hub model which is officially pretrained. See List of Pretrained Models for the complete list of pretrained models on TF-Hub. When initializing from a pretrained model, the encoder architecture of the pretrained model will be used and the encoder architecture you set in the config (`configs/models/bert_en_uncased_base.yaml` in this case) will be ignored.

You can change `--mode=train` to `--mode=train_and_eval` if you want to see evaluation results. But you need to specify the path to the evaluation data by setting `task.validation_data.input_path` in `PARAMS`.

Run on Cloud TPUs

Next, we will describe how to run the train.py on Cloud TPUs.

Setup

First, you need to create a `tf-nightly` TPU with ctpu tool:

```
export TPU_NAME=YOUR_TPU_NAME
ctpu up -name $TPU_NAME --tf-version=nightly --tpu-size=YOUR_TPU_SIZE --project=YOUR_PROJECT
```

and then install Model Garden and required dependencies:

```
git clone https://github.com/tensorflow/models.git
export PYTHONPATH=$PYTHONPATH:/path/to/models
pip3 install --user -r official/requirements.txt
```

Fine-tuning Sentence Classification with BERT from TF-Hub

This example fine-tunes BERT-base from TF-Hub on the the Multi-Genre Natural Language Inference (MultiNLI) corpus using TPUs.

Firstly, you can prepare the fine-tuning data using `create_finetuning_data.py` script. For GLUE tasks, you can (1) download the GLUE data by running this script and unpack it to some directory `$GLUE_DIR`, (2) prepare the vocabulary file, and (3) run the following command:

```
export GLUE_DIR=~/glue
export VOCAB_FILE=~/uncased_L-12_H-768_A-12/vocab.txt

export TASK_NAME=MNLI
```

```
export OUTPUT_DATA_DIR=gs://some_bucket/datasets
python3 data/create_finetuning_data.py \
 --input_data_dir=${GLUE_DIR}/${TASK_NAME}/ \
 --vocab_file=${VOCAB_FILE} \
 --train_data_output_path=${OUTPUT_DATA_DIR}/${TASK_NAME}_train.tf_record \
 --eval_data_output_path=${OUTPUT_DATA_DIR}/${TASK_NAME}_eval.tf_record \
 --meta_data_file_path=${OUTPUT_DATA_DIR}/${TASK_NAME}_meta_data \
 --fine_tuning_task_type=classification --max_seq_length=128 \
 --classification_task_name=${TASK_NAME}
```

Resulting training and evaluation datasets in `tf_record` format will be later passed to train.py. We will support to read dataset from tensorflow_datasets (TFDS) and use tf.text for pre-processing soon.

Then you can execute the following commands to start the training and evaluation job.

```
export INPUT_DATA_DIR=gs://some_bucket/datasets
export OUTPUT_DIR=gs://some_bucket/my_output_dir

# See tfhub BERT collection for more tfhub models:
# https://tfhub.dev/google/collections/bert/1
export BERT_HUB_URL=https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3

# Override the configurations by FLAGS. Alternatively, you can directly edit
# `configs/experiments/glue_mnli_matched.yaml` to specify corresponding fields.
export PARAMS=task.train_data.input_path=$INPUT_DATA_DIR/mnli_train.tf_record
export PARAMS=$PARAMS,task.validation_data.input_path=$INPUT_DATA_DIR/mnli_eval.tf_record
export PARAMS=$PARAMS,task.hub_module_url=$BERT_HUB_URL
export PARAMS=$PARAMS,runtime.distribution_strategy=tpu

python3 train.py \
 --experiment=bert/sentence_prediction \
 --mode=train_and_eval \
 --model_dir=$OUTPUT_DIR \
 --config_file=configs/models/bert_en_uncased_base.yaml \
 --config_file=configs/experiments/glue_mnli_matched.yaml \
 --tfhub_cache_dir=$OUTPUT_DIR/hub_cache \
 --tpu=${TPU_NAME} \
 --params_override=$PARAMS
```

You can monitor the training progress in the console and find the output models in `$OUTPUT_DIR`.

Fine-tuning SQuAD with a pre-trained BERT checkpoint

This example fine-tunes a pre-trained BERT checkpoint on the Stanford Question Answering Dataset (SQuAD) using TPUs. The SQuAD website contains detailed information about the SQuAD datasets and evaluation. After downloading the SQuAD datasets and the pre-trained BERT checkpoints, you can run the following command to prepare the `tf_record` files:

```
export SQUAD_DIR=~/squad
export BERT_DIR=~/uncased_L-12_H-768_A-12
export OUTPUT_DATA_DIR=gs://some_bucket/datasets

python3 create_finetuning_data.py \
 --squad_data_file=${SQUAD_DIR}/train-v1.1.json \
 --vocab_file=${BERT_DIR}/vocab.txt \
 --train_data_output_path=${OUTPUT_DATA_DIR}/train.tf_record \
 --meta_data_file_path=${OUTPUT_DATA_DIR}/squad_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`.

Then, you can start the training and evaluation jobs:

```
export SQUAD_DIR=~/squad
export INPUT_DATA_DIR=gs://some_bucket/datasets
export OUTPUT_DIR=gs://some_bucket/my_output_dir

# See the following link for more pre-trained checkpoints:
# https://github.com/tensorflow/models/blob/master/official/nlp/docs/pretrained_models.md
export BERT_DIR=~/uncased_L-12_H-768_A-12

# Override the configurations by FLAGS. Alternatively, you can directly edit
# `configs/experiments/squad_v1.1.yaml` to specify corresponding fields.
# Also note that the training data is the pre-processed tf_record file, while
# the validation file is the raw json file.
export PARAMS=task.train_data.input_path=$INPUT_DATA_DIR/train.tf_record
export PARAMS=$PARAMS,task.validation_data.input_path=$SQUAD_DIR/dev-v1.1.json
export PARAMS=$PARAMS,task.validation_data.vocab_file=$BERT_DIR/vocab.txt
export PARAMS=$PARAMS,task.init_checkpoint=$BERT_DIR/bert_model.ckpt
export PARAMS=$PARAMS,runtime.distribution_strategy=tpu

python3 train.py \
 --experiment=bert/squad \
 --mode=train_and_eval \
 --model_dir=$OUTPUT_DIR \
 --config_file=configs/models/bert_en_uncased_base.yaml \
 --config_file=configs/experiments/squad_v1.1.yaml \
 --tpu=${TPU_NAME} \
 --params_override=$PARAMS
```

Note: More examples about pre-training will come soon.