Training and Evaluation with TensorFlow 1

Python 3.6 TensorFlow 1.15

This page walks through the steps required to train an object detection model. It assumes the reader has completed the following prerequisites:

1. The TensorFlow Object Detection API has been installed as documented in the installation instructions.
2. A valid data set has been created. See this page for instructions on how to generate a dataset for the PASCAL VOC challenge or the Oxford-IIIT Pet dataset.

Recommended Directory Structure for Training and Evaluation

```
.
├── data/
│   ├── eval-00000-of-00001.tfrecord
│   ├── label_map.txt
│   ├── train-00000-of-00002.tfrecord
│   └── train-00001-of-00002.tfrecord
└── models/
    └── my_model_dir/
        ├── eval/                 # Created by evaluation job.
        ├── my_model.config
        └── train/                #
            └── model_ckpt-100-data@1  # Created by training job.
            └── model_ckpt-100-index   #
            └── checkpoint             #
```

Writing a model configuration

Please refer to sample TF1 configs and configuring jobs to create a model config.

Model Parameter Initialization

While optional, it is highly recommended that users utilize classification or object detection checkpoints. Training an object detector from scratch can take days. To speed up the training process, it is recommended that users re-use the feature extractor parameters from a pre-existing image classification or object detection checkpoint.
The `train_config` section in the config provides two fields to specify pre-existing checkpoints:

- `fine_tune_checkpoint` : a path prefix to the pre-existing checkpoint (ie:"/usr/home/username/checkpoint/model.ckpt-#####").
- `fine_tune_checkpoint_type` : with value `classification` or `detection` depending on the type.

A list of detection checkpoints can be found here.

Local

Training

A local training job can be run with the following command:

```
# From the tensorflow/models/research/ directory
PIPELINE_CONFIG_PATH={path to pipeline config file}
MODEL_DIR={path to model directory}
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
python object_detection/model_main.py \
    --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
    --model_dir=${MODEL_DIR} \
    --num_train_steps=${NUM_TRAIN_STEPS} \
    --sample_1_of_n_eval_examples=${SAMPLE_1_OF_N_EVAL_EXAMPLES} \
    --alsologtostderr
```

where `${PIPELINE_CONFIG_PATH}` points to the pipeline config and `${MODEL_DIR}` points to the directory in which training checkpoints and events will be written. Note that this binary will interleave both training and evaluation.

Google Cloud AI Platform

The TensorFlow Object Detection API supports training on Google Cloud AI Platform. This section documents instructions on how to train and evaluate your model using Cloud AI Platform. The reader should complete the following prerequistes:

1. The reader has created and configured a project on Google Cloud AI Platform. See Using GPUs and Using TPUs guides.
2. The reader has a valid data set and stored it in a Google Cloud Storage bucket. See this page for instructions on how to generate a dataset for the PASCAL VOC challenge or the Oxford-IIIT Pet dataset.

Additionally, it is recommended users test their job by running training and evaluation jobs for a few iterations locally on their own machines.

Training with multiple workers with single GPU

Google Cloud ML requires a YAML configuration file for a multiworker training job using GPUs. A sample YAML file is given below:

```
trainingInput:
  runtimeVersion: "1.15"
  scaleTier: CUSTOM
  masterType: standard_gpu
  workerCount: 9
  workerType: standard_gpu
  parameterServerCount: 3
  parameterServerType: standard
```

Please keep the following guidelines in mind when writing the YAML configuration:

- A job with n workers will have n + 1 training machines (n workers + 1 master).

- The number of parameters servers used should be an odd number to prevent a parameter server from storing only weight variables or only bias variables (due to round robin parameter scheduling).
- The learning rate in the training config should be decreased when using a larger number of workers. Some experimentation is required to find the optimal learning rate.

The YAML file should be saved on the local machine (not on GCP). Once it has been written, a user can start a training job on Cloud ML Engine using the following command:

```
# From the tensorflow/models/research/ directory
cp object_detection/packages/tf1/setup.py .
gcloud ml-engine jobs submit training object_detection_`date +%m_%d_%Y_%H_%M_%S` \
    --runtime-version 1.15 \
    --python-version 3.6 \
    --job-dir=gs://${MODEL_DIR} \
    --package-path ./object_detection \
    --module-name object_detection.model_main \
    --region us-central1 \
    --config ${PATH_TO_LOCAL_YAML_FILE} \
    -- \
    --model_dir=gs://${MODEL_DIR} \
    --pipeline_config_path=gs://${PIPELINE_CONFIG_PATH}
```

Where `${PATH_TO_LOCAL_YAML_FILE}` is the local path to the YAML configuration, `gs://${MODEL_DIR}` specifies the directory on Google Cloud Storage where the training checkpoints and events will be written to and `gs://${PIPELINE_CONFIG_PATH}` points to the pipeline configuration stored on Google Cloud Storage.

Users can monitor the progress of their training job on the ML Engine Dashboard.

Training with TPU

Launching a training job with a TPU compatible pipeline config requires using a similar command:

```
# From the tensorflow/models/research/ directory
cp object_detection/packages/tf1/setup.py .
gcloud ml-engine jobs submit training `whoami`_object_detection_`date
+%m_%d_%Y_%H_%M_%S` \
    --job-dir=gs://${MODEL_DIR} \
    --package-path ./object_detection \
    --module-name object_detection.model_tpu_main \
    --runtime-version 1.15 \
    --python-version 3.6 \
    --scale-tier BASIC_TPU \
    --region us-central1 \
    -- \
    --tpu_zone us-central1 \
    --model_dir=gs://${MODEL_DIR} \
    --pipeline_config_path=gs://${PIPELINE_CONFIG_PATH}
```

In contrast with the GPU training command, there is no need to specify a YAML file, and we point to the *object_detection.model_tpu_main* binary instead of *object_detection.model_main*. We must also now set `scale-`

`tier` to be `BASIC_TPU` and provide a `tpu_zone`. Finally as before `pipeline_config_path` points to a points to the pipeline configuration stored on Google Cloud Storage (but is now must be a TPU compatible model).

Evaluation with GPU

Note: You only need to do this when using TPU for training, as it does not interleave evaluation during training, as in the case of Multiworker GPU training.

Evaluation jobs run on a single machine, so it is not necessary to write a YAML configuration for evaluation. Run the following command to start the evaluation job:

```
# From the tensorflow/models/research/ directory
cp object_detection/packages/tf1/setup.py .
gcloud ml-engine jobs submit training object_detection_eval_`date
+%m_%d_%Y_%H_%M_%S` \
    --runtime-version 1.15 \
    --python-version 3.6 \
    --job-dir=gs://${MODEL_DIR} \
    --package-path ./object_detection \
    --module-name object_detection.model_main \
    --region us-central1 \
    --scale-tier BASIC_GPU \
    -- \
    --model_dir=gs://${MODEL_DIR} \
    --pipeline_config_path=gs://${PIPELINE_CONFIG_PATH} \
    --checkpoint_dir=gs://${MODEL_DIR}
```

Where `gs://${MODEL_DIR}` points to the directory on Google Cloud Storage where training checkpoints are saved (same as the training job), as well as to where evaluation events will be saved on Google Cloud Storage and `gs://${PIPELINE_CONFIG_PATH}` points to where the pipeline configuration is stored on Google Cloud Storage.

Typically one starts an evaluation job concurrently with the training job. Note that we do not support running evaluation on TPU, so the above command line for launching evaluation jobs is the same whether you are training on GPU or TPU.

Running Tensorboard

Progress for training and eval jobs can be inspected using Tensorboard. If using the recommended directory structure, Tensorboard can be run using the following command:

```
tensorboard --logdir=${MODEL_DIR}
```

where `${MODEL_DIR}` points to the directory that contains the train and eval directories. Please note it may take Tensorboard a couple minutes to populate with data.