TF Model Garden Ranking Models

Overview

This is an implementation of DLRM and DCN v2 ranking models that can be used for tasks such as CTR prediction.

The model inputs are numerical and categorical features, and output is a scalar (for example click probability). The model can be trained and evaluated on GPU, TPU and CPU. The deep ranking models are both memory intensive (for embedding tables/lookup) and compute intensive for deep networks (MLPs). CPUs are best suited for large sparse embedding lookup, GPUs for fast compute. TPUs are designed for both.

When training on TPUs we use TPUEmbedding layer for categorical features. TPU embedding supports large embedding tables with fast lookup, the size of embedding tables scales linearly with the size of TPU pod. We can have up to 90 GB embedding tables for TPU v3-8 and 5.6 TB for v3-512 and 22,4 TB for TPU Pod v3-2048.

The Model code is in TensorFlow Recommenders library, while input pipeline, configuration and training loop is here.

Prerequisites

To get started, download the code from TensorFlow models GitHub repository or use the pre-installed Google Cloud VM.

```
git clone https://github.com/tensorflow/models.git
export PYTHONPATH=$PYTHONPATH:$(pwd)/models
```

We also need to install TensorFlow Recommenders library. If you are using tf-nightly make sure to install tensorflow-recommenders without its dependancies by passing `--no-deps` argument.

For tf-nightly:

```
pip install tensorflow-recommenders --no-deps
```

For stable TensorFlow 2.4+ releases:

```
pip install tensorflow-recommenders
```

Dataset

The models can be trained on various datasets, Two commonly used ones are Criteo Terabyte and Criteo Kaggle datasets. We can train on synthetic data, by setting the flag `use_synthetic_data=True`.

Download

The dataset is the Terabyte click logs dataset provided by Criteo. Follow the instructions at the Criteo website to download the data.

Note that the dataset is large (~1TB).

Preprocess the data

Follow the instructions in Data Preprocessing to preprocess the Criteo Terabyte dataset.

Data preprocessing steps are summarized below.

Integer feature processing steps, sequentially:

1. Missing values are replaced with zeros.
2. Negative values are replaced with zeros.
3. Integer features are transformed by log(x+1) and are hence tf.float32.

Categorical features:

1. Categorical data is bucketized to tf.int32.
2. Optionally, the resulting integers are hashed to a lower dimensionality. This is necessary to reduce the sizes of the large tables. Simple hashing function such as modulus will suffice, i.e. feature_value % MAX_INDEX.

The vocabulary sizes resulting from pre-processing are passed in to the model trainer using the model.vocab_sizes config. Note that provided values in sample below are only valid for Criteo Terabyte dataset.

The full dataset is composed of 24 directories. Partition the data into training and eval sets, for example days 1-23 for training and day 24 for evaluation.

Training and eval datasets are expected to be saved in many tab-separated values (TSV) files in the following format: numberical fetures, categorical features and label.

On each row of the TSV file, the first one is the label (either 0 or 1), the next `num_dense_features` inputs are numerical features, then `vocab_sizes` categorical features. Each i-th categorical feature is expected to be an integer in the range of `[0, vocab_sizes[i])`.

Train and Evaluate

To train DLRM model we use dot product feature interaction, i.e. `interaction: 'dot'` to train DCN v2 model we use `interaction: 'cross'`.

Training on TPU

```
export TPU_NAME=my-dlrm-tpu
export EXPERIMENT_NAME=my_experiment_name
export BUCKET_NAME="gs://my_dlrm_bucket"
export DATA_DIR="${BUCKET_NAME}/data"
export EMBEDDING_DIM=32

python3 models/official/recommendation/ranking/train.py --mode=train_and_eval \
--model_dir=${BUCKET_NAME}/model_dirs/${EXPERIMENT_NAME} --params_override="
runtime:
    distribution_strategy: 'tpu'
task:
    use_synthetic_data: false
    train_data:
        input_path: '${DATA_DIR}/train/*'
        global_batch_size: 16384
    validation_data:
        input_path: '${DATA_DIR}/eval/*'
        global_batch_size: 16384
    model:
```

```
        num_dense_features: 13
        bottom_mlp: [512,256,${EMBEDDING_DIM}]
        embedding_dim: ${EMBEDDING_DIM}
        top_mlp: [1024,1024,512,256,1]
        interaction: 'dot'
        vocab_sizes: [39884406, 39043, 17289, 7420, 20263, 3, 7120, 1543, 63,
            38532951, 2953546, 403346, 10, 2208, 11938, 155, 4, 976, 14,
            39979771, 25641295, 39664984, 585935, 12972, 108, 36]
trainer:
    use_orbit: true
    validation_interval: 85352
    checkpoint_interval: 85352
    validation_steps: 5440
    train_steps: 256054
    steps_per_loop: 1000
"
```

The data directory should have two subdirectories:

- $DATA_DIR/train
- $DATA_DIR/eval

Training on GPU

Training on GPUs are similar to TPU training. Only distribution strategy needs to be updated and number of GPUs provided (for 4 GPUs):

```
export EMBEDDING_DIM=8

python3 official/recommendation/ranking/train.py --mode=train_and_eval \
--model_dir=${BUCKET_NAME}/model_dirs/${EXPERIMENT_NAME} --params_override="
runtime:
  distribution_strategy: 'mirrored'
  num_gpus: 4
...
"
```