598CM - Individual Project: Learned Cost Model for TPUs

Fall 2023

1 Logistics

Due: November 24th, 2023
Group Size: individual
Submission: Please read the last section.

2 Introduction

In this project you will a train machine-learning model to predict the best tile size selection for kernels of full tensor programs, represented as computational graphs. You will be given a *production-quality* dataset from Google. We will use a model close to what is suggested in [1] paper as the baseline. We will release a full grading scheme closer to the deadline with the baseline scores. There *won't* be any competitive grading.

3 The Dataset

In this section, we will provide an overview of the parts of the dataset that are relevant to this project. For more information on the dataset, please consult the original paper [2]. It is worth mentioning that this is a production-quality dataset that has been used by Google and which got open-sourced recently.

The dataset, TpuGraphs [2], is a performance prediction dataset on full tensor programs, represented as computational graphs. Each graph represents the main computation of an ML program, which is usually one or many training steps or one inference step. The graphs in the dataset are collected from open-source ML programs, featuring popular models (e.g., ResNet, EfficientNet, Mask R-CNN, and a large variety of Transformer) for a wide range of tasks, e.g., vision, NLP, speech, audio, recommendation, and generative AI. Each data sample contains a computational graph, a compilation configuration, and the execution time when executing the graph when compiled with the given configuration on a Tensor Processing Unit (TPU) v3 [39], an accelerator for ML workloads [2].

3.1 Tile-Size Selection

In this project, we will only use the subset of the dataset that contains compilation configurations for tile-size selection, i.e., tile configurations. A tile configuration controls the tile size of each fused subgraph.

Each node in a tensor computation graph represents a tensor operation, such as matrix multiplication, convolution, element-wise addition, etc. A kernel, represented as a fused subgraph, is then a fusion of multiple tensor operations. For example, Convolution-BatchNorm is a common fused kernel that appears in Convolutional Neural Networks. The most important optimization at the kernel level is tile size selection: selecting the shape of a tile of the output tensor to maximize compute efficiency of the hardware,

while the required regions of input, output, and intermediate data fit in the local cache or scratchpad memory

4 A Quickstart on the Infrastructure and Baseline Models

The original authors have open-sourced a repository with some baseline models and a relevant helpful infrastructure. These should help you get started and they will also allow you to create the deliverables required for grading easily.

Note that this infrastructure is under early development and thus, somewhat sensitive, so *make sure that you follow the steps exactly*! This includes making sure that you use the same paths that we use. Additionally, if you encounter a bug, please let us know and we can file a bug report (or a pull request if we fix it) together.

4.1 Clone the Repository

Clone the repository by running:

```
git clone https://github.com/google-research-datasets/tpu_graphs
```

It does *not* matter where you will clone the repository. But we will need to refer to the root path of the repository again, and we will refer to it as `ROOT_DIR`.

4.2 Download the Dataset

Before following the steps below, make sure that there is no directory `~/data`. The following step will create this directory and download the dataset there.

Now, navigate to `ROOT_DIR` and run:

```
python3 echo_download_commands.py | bash
```

You should be able to find the three following files:

- `~/data/tpugraphs/npz_tile_xla_test.tar`
- `~/data/tpugraphs/npz_tile_xla_train.tar`
- `~/data/tpugraphs/npz_tile_xla_valid.tar`

4.3 Create a Conda Environment

First, create a conda environment as follows:

```
conda create -n tpugraphs python=3.10
conda activate tpugraphs

conda install -c conda-forge tensorflow
conda install -c conda-forge tqdm

pip install tensorflow_gnn --pre
pip install tensorflow-ranking
conda clean --all
```

4.4 Train a Baseline Model

Before proceeding, make sure that there is no directory `~/out`. The following step will create this directory and use it to store some files.

We will first train a baseline model with a subset of the dataset (i.e., toy data). Navigate to `ROOT_DIR` and run:

```
python tiles_train.py --model=MLP --toy_data=True
```

After this completes, you should see the directory `~/out/tpugraphs_tiles`. In it, you should see (at least) the following 3 entries:

1. `model_<hash>` (this is a directory)
2. `run_<hash>.jsonz`
3. `results_<timestamp>.csv`

The last file will not concern us for now.

The `<hash>` will be some hex hash like: `model_d691d986e1e5414ba622691b0a7c5c05`. The two hashes in the two files should match. Every time you train a model, you should see a new such pair of files. The most important of the two is the `model_<hash>` directory because it contains the (weights for the) *trained model*. But, you still need the matching `run_<hash>.jsonz`, so do not delete that.

4.4.1 Full Dataset

Unfortunately, you cannot get the model to output predictions if it is trained on the toy dataset [1]. Training on the toy dataset is used only for debugging purposes. So, you need to train on the full dataset. You can do that just by *not* providing the `toy_data` argument, as in the example below:

```
python tiles_train.py --model=MLP
```

This should be fairly quick given that we're using an MLP.

5 Evaluating your Model

5.1 Generating predictions over the test set

The goal of this step is to run the model over the test set and produce a CSV that has the top 5 predictions of the model for each kernel. The CSV will look like this:

```
ID,TopConfigs
tile:xla:57612df6091fe11c498a48fc069186dc,207;103;161;233;205
tile:xla:89f5c24cf4c3c6d65ffff9c207adc20d,261;221;564;1361;325
tile:xla:7d314295403544719fc8fb837cd3a953,183;815;889;1225;229
tile:xla:724dbd338923da4d9fa688d5b19a66fc,130;660;2142;1756;1835
```

The `ID` column is the ID of the kernel and the `TopConfigs` column contains exactly 5 numbers for each ID, separated by `;`, which are the top 5 choices of the model for this kernel.

[1] See this issue if you want more information.

Every time you invoke `tiles_train.py`[2], it produces a file `results_<timestamp>.csv`, which contains the predictions of the trained model. However, we will assume that you don't have such a file and you only have the trained model.

We created a script that, given a model directory (see previous section), creates a CSV with the predictions. Clone the following repository:

```
git clone https://github.com/ADAPT-uiuc/tpu_graphs_scripts
```

And copy *all* of its contents to `ROOT_DIR`. Then, navigate to `ROOT_DIR` and invoke the following script:

```
python tiles_gen_predictions_csv.py --dir ~/out/tpugraphs_tiles/model_<hash>
```

In the `-dir` argument, you should provide the path of the model. If you have been following us, it should look similar to the above. This script should create a file `predictions_output_model.csv` (which is similar to `results_<timestamp>.csv` we mentioned above).

Finally, run the following script to get a CSV ready for submission to Kaggle (see next section; this script just inserts dummy values for the layout task):

```
python create_kaggle_csv.py
```

This assumes that you have an input CSV file named `predictions_output_model.csv` in the same directory. If you already have a different CSV (e.g., by running `tiles_train.py`), then skip the invocation of `tiles_gen_predictions_csv.py` and just invoke:

```
python create_kaggle_csv.py --inputcsv path/to/csv
```

5.2 Get a Kaggle score

There is a Kaggle competition over the test set of this dataset, where you upload your predictions and you get a score. The highest score wins. We will evaluate (and grade) your model using this Kaggle score. So, note that you will get graded based on how your model performs on the *real* test set, for which even we do not have the labels.

You can follow this link: https://www.kaggle.com/competitions/predict-ai-model-runtime to view the competition. To get a score, make a Kaggle account, review and accept the rules of the competition and submit your CSV. This competition involves one other task on layout selection, but the CSV that our script outputs already has dummy values for these entries so you do not need to worry about that.

Please note that you can make **up to 3 submissions per day**.

6 A Closer Look at the Script that Generates Predictions

It is necessary for full marks that your model works with `tiles_gen_predictions_csv.py`, as we outlined above. So, we provide a small description of what this script does.

The `main()` function roughly does the following things, in order:

1. Load the test set (link)
2. Load the Keras model (link)

[2]Please refer to the previous section for more information on `tiles_train.py`

3. Find out what model class this model uses and instantiate it (link)
4. Copy the trainable variables from the Keras model (link)
5. Run the forward pass on all the graphs of the dataset and generate predictions/ranks (link)

For step 2), there is a list of predefined models here that you can use to get started. You can also train any of them with `tiles_train.py`. For example, earlier we used the `MLP` model because it's the fastest, but you can e.g., use `EarlyJoinSAGE` as follows:

```
python tiles_train.py --model=EarlyJoinSAGE
```

Theoretically, it does not matter how you train your model as long as you can plug it into the script. But in practice, it will be easier if you start by modifying one of the predefined models, making sure that during the process your model outputs the same things as the predefined models, because in that way you can: (a) train your model using `tiles_train.py` and so, also (b) plug it into our script effortlessly.

7 What You Should Turn In

You should turn in the following:

1. A modified version of `baselines/tiles/models.py` (i.e., this file) with *your model class*! This can be a modification of the predefined models or a new class. Note that technically, we allow you to submit this file unmodified. But this baseline is pretty bad.
2. A directory containing your model, similar to the `model_<hash>` directory we saw above, *as well as* the matching `run_<hash>.jsonz` file.

We should be able to swap `models.py` with your file (number 1) above), and then run `tiles_gen_predictions_csv.py` as mentioned above.

References

[1] S. J. Kaufman, P. M. Phothilimthana, Y. Zhou, C. Mendis, S. Roy, A. Sabne, and M. Burrows. A learned performance model for tensor processing units. In *Conference on Machine Learning and Systems*, 2021.

[2] P. M. Phothilimthana, S. Abu-El-Haija, K. Cao, B. Fatemi, C. Mendis, and B. Perozzi. Tpugraphs: A performance prediction dataset on large tensor computational graphs, 2023.