High Performance Machine Learning
Lab 4

Paweł Rościszewski
Department of Computer Architecture
Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology

Introduction

In this lab we will modify the TesorFlow-based training program developed in previous labs.
Note: If someone introduced custom modifications to the code and wants to keep them, they can continue with their version. A version with checkpointing might be more fun, because it will probably generate interesting images.

1 Task 1: Compare hardware performance (4 points)

Let's compare performance of the training program depending on the used hardware. In this task we will use the performance measurement hook implemented in Lab2.

To check where the individual variables and operations are placed by the TensorFlow placer, set he log_device_placement parameter of the session config. As a result, you should see device placement info in the **Jupyter console** (not in the Jupyter cells) during the training.

To modify placement of the operations, use the device_fn parameter of the estimator RunConfig[1].

Make sure that proper computing devices are utilized using the *top* and *nvidia-smi* diagnostic tools. This way, we can measure performance of the CPU-only version.

To pass the task: show the application running only on CPU, compare its performance to the initial version, discuss.

[1] https://www.tensorflow.org/api_docs/python/tf/estimator/RunConfig

2 Task 2: Converting Jupyter notebooks (2 points)

Jupyter notebooks are very useful for interactive code development, especially in the context of visualization of data and intermediate results. However, for running the code remotely on a set of distributed hosts, plain Python scripts can be more useful. The Jupyter nbconvert script allows to convert the Jupyter notebook to a python script:

```
jupyter-nbconvert HPMLLab.ipynb --to python
```

Note: In order to work properly, the exported code may require manual improvements.

To pass the task: report a screenshot of a working standalone training script.

3 Task 3: Convert data provider to pure tf.data.Dataset (4 points)

In Lab5 we will use the TF distribute strategies[2] to perform a distributed multi-node training. Because the tf.distribute.Strategy API handles dividing batches across nodes, it requires data providers to return instances tf.data.Dataset. Unfortunately, the StarGAN implementation used in our lab does not support this. However, since we are High Performance Machine Learning professionals, we can fix it.

One way to achieve that is to get rid of the provide_data function, and use a modified version of the provide_dataset function. Since the GAN model requires the input in the form of lists of tensors (input_data = [image1, image2, ..., imageN], input_data_domain_label = [label1, label2, ..., labelN]), we have to modify the TFGAN library to properly process input provided by our tf.data.Dataset. The changes are required in the _model_fn function used by StarGANEstimator, which we can find in our virtual environment[3].

Setting the input_data and input_data_domain_label parameters has to be matched with the provide_dataset function in modes different than PREDICT.
Hint: Remember that a changed imported library has to be reloaded.

To pass the task: show that the training works with tf.data.Dataset.

[2] https://www.tensorflow.org/guide/distributed_training
[3] venv_directory/lib/python3.6/site-packages/tensorflow_gan/python/estimator/stargan_estimator.py