High Performance Machine Learning
Lab 2

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

1 Introduction

The main functionality of a training program in machine learning is executing the training operations, which in TensorFlow are represented by the computational graph and actually executed within estimator train() calls. Still, there are many additional functionalities of a training program that have to be performed once in a while, for example saving checkpoints, logging and monitoring progress, computing validation loss, modifying certain training parameters depending on various advanced training methods etc.

In the training code investigated in previous labs, such a functionality was plotting the summary images. It required additional code in the training loop. In many practical training programs, the number of such additional functionalities is very high and the training loop becomes complicated, unreadable and hard to maintain. The SessionRunHook mechanism comes handy to write the training program in a more elegant way through building it from hook components.

In this lab we will practice using, extending and writing own SessionRunHook implementations on the example of the training program investigated in previous labs. We will also improve the implementation by removing a bug and examine the performance depending on different batch sizes.

2 Task 1: Get rid of the training loop (2 points)

Although the training loop in our code performs training steps until a given *max number of steps*, notice that the *cur_step* counter is not being saved in the checkpoints. The counter is set to 0 each time the training is restarted, which would actually result in performing more steps than required.

The estimator API uses the *global_step* variable and saves it within the checkpoints. It is automatically initialized when reading from a checkpoint. This means, that we can get rid of the training loop, keep a single call to the esti-

mator train method, and use the *max_steps* parameter instead of *steps* to make sure that the training in total completes the correct number of steps.
Note: getting rid of the training loop will require removing the summary image plotting. Don't worry, we will take care of that later.

To pass the task: show that after reading from a checkpoint, the estimator train method performs the correct number of steps.

3 Task 2: Re-implement the logging functionality using LoggingTensorHook (3 points)

TensorFlow comes with a set of SessionRunHook implementations called 'predefined' hooks[1] for chosen frequently used functionalities. Some of them were already used in Lab1 under the hood of the estimator API. For example, setting the *model_dir* argument resulted in creating a *CheckpointSaverHook*, and setting the *add_summaries* argument resulted in creating a *SummarySaverHook*.

Hooks can be also added explicitly using the *hooks* argument to the estimator *train()* method.

Another 'pre-defined' SessionRunHook is the LoggingTensorHook[2] for logging the values of chosen variables in given step (*every_n_iter*) or second (*every_n_step*) periods. A list of tensors which should be evaluated for the logging purposes during the appropriate iterations is given as the *tensors* argument. Optionally, a callback function can be provided through the *formatter* argument. It is a function that as argument accepts a map where keys are tensors and values are current values of these tensors. In this task, let's use the LoggingTensorHook along with a custom formatter to create a hook component which will replace the logging block in the training loop.

To pass the task: show that the plotting the summary images based on LoggingTensorHook works.

4 Task 3: Implement SessionRunHook for performance logging (3 points)

The hooks that were used and extended in previous tasks are called SessionRunHooks because they all implement the *tf.train.SessionRunHook*[3] programming interface. The interface specifies methods called at specific moments in the lifecycle of the training including *after_run()* method called after each training iteration.

The SessionRunHook interface can be used to implement and add a custom functionality to the training program. Let's practice this by extending out

[1]https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/training/session_run_hook.py
[2]https://www.tensorflow.org/api_docs/python/tf/estimator/LoggingTensorHook
[3]https://www.tensorflow.org/api_docs/python/tf/estimator/SessionRunHook

training program with a custom PerformanceMeasurementHook which will, once a given number of steps, save the current training step and time and print the training speed measured by number of *images* per second.

Hint: The new class implementing the tf.train.SessionRunHook can be provided with specific parameters, such as the number of steps in the measurement period or the batch_size using the __init__() initializer method.

To pass the task: show that the custom hook for performance measurement works, discuss.

5 Task 4: Find the best batch size in terms of performance (2 points)

The experiments in Lab1 revealed that our original training code leaves a certain amount of free space in the GPU memory. This might allow us to increase the number of images used in each training iteration (batch_size). Use the performance measurement tool implemented within this lab to examine the performance of various batch_size settings.

To pass the task: discuss the performance results for chosen batch sizes and report the best achieved value.