Lab Course: Distributed Data Analytics
Exercise Sheet 9

Prof. Dr. Dr. Schmidt-Thieme, Daniela Thyssens
Information Systems and Machine Learning Lab
University of Hildesheim
Submission deadline: Sunday July 10, 23:59PM (on LearnWeb, course code: 3116)

Instructions

Please following these instructions for solving and submitting the exercise sheet.

1. You should submit two items a) a zipped file containing python scripts and b) a pdf document.
2. In the pdf document you will explain your approach (i.e. how you solved a given problem), and present your results.
3. The submission needs to be made before the deadline, only through learnweb.
4. Unless explicitly stated, you are not allowed to use scikit, sklearn or any other library for solving any part. All implementations must be done by yourself.

Implementing Parallel Stochastic Gradient Descent (10 points)

For this lab, you will implement the network following architecture and use the MNIST dataset.

1. conv1: convolution
2. pool1: max pooling
3. rectified linear activation (RELU)
4. conv2: convolution
5. pool2: max pooling
6. rectified linear activation (RELU)
7. dropout
8. pool3: max pooling
9. FC1: fully connected layer
10. rectified linear activation (RELU)
11. FC2: fully connected layer
12. softmax_layer: final output predictions i.e. classify into one of the ten classes.

First having looked at the design of the network and then its regularization, we are now ready to bring our knowledge full circle by going back to where we started in this semester, MPI. PSGD makes use of the simplicity of SGD along with the work division strategies we have learned this semester to arrive at a parallel algorithm that is aimed at speeding up the training of our model. For a detailed description of SGD please refer to the lecture from Lars which can be found on learn web. For this lab, we will be implementing PSGD using pytorch. The idea behind PSGD is to run SGD on a smaller partition of the whole dataset on P many workers and aggregate the learned weights of the model at the end. Steps involved in PSGD can be found on slide 6. Pytorch allows us to access the model weights using the following command "model.named_parameters()".

This command will provide an iterator which can be used to access each weight and bias of the network as shown below:

```
for name, param in model.named_parameters():
        print(name, param)
```

Using the MPI framework, you are required to implement the algorithm on slide 6 of the SGD lecture (from learnweb). Line 3 of the algorithm should be paid special attention to since it is the clue for how the different workers will behave. Each worker needs to calculate the weights until either the termination condition of the max iteration (you can choose how you want to design the training algorithm). Please report the timings of the vanilla SGD ($P = 1$) and for $P = 2, 3, 4, 5$. Show with the help of a speedup graph how adding workers impacts the time needed for PSGD. Please also report on the accuracy of the model for ($P = 1$) and for $P = 2, 3, 4, 5$. Comment on the behavior you see.

PyTorch distributed execution (10 points)

For this exercise, we will now let PyTorch handle the multiprocessing nature of the execution for us. You are required to implement HogWild SGD but this time using the torch.multiprocessing library. We will be using a share memory architecture here for a comprehensive explanation of shared memory models/archs and multiprocessing in PyTorch please refer to the annex. The algorithm for HogWild can be found on slide 12 of the lecture (on learnweb). Please note that by letting PyTorch handle the multiprocessing and using a shared-memory architecture, you do not have to manually implement lines 7 to 9. Similar to the previous exercise, please report your speedup graphs and your final accuracies for the models and comment on any discrepancies you notice.

Annex

1. Torch Multiprocessing `https://pytorch.org/docs/stable/multiprocessing.html`
2. Accessing Weights of a model `https://discuss.pytorch.org/t/access-all-weights-of-a-model/77672`
3. Interesting read `https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-`
4. Shared Memory HogWild in Pytorch `https://pytorch.org/docs/stable/notes/multiprocessing.html`