ECE-GY 9143 - High Performance Machine Learning

# Homework Assignment 5

Parijat Dube and Kaoutar El Maghraoui

Due Date: April 24 | Spring 2024 | Max Points: 100

**Instructions:**
This lab is intended to be performed **individually**, great care will be taken in verifying that students are authors of their own submission.

*This lab has two parts. Part-A on distributed deep learning and Part-B on Quantization.* You can do this lab on NYU HPC or Google colab where you will have access to multiple GPUs (up to 4).

# Part-A: Distributed deep learning (50 points)

Distributed deep learning [1] is the de facto approach to training neural networks at scale. Synchronous SGD (SSGD) has been the most widely used training algorithms among all the learning algorithms. In this lab, we are going to experiment with PyTorch's DataParallel Module [2], which is PyTorch's SSGD implementation across a number of GPUs on the same server. In particular, we re-use lab 2 code with **default SGD solver and its hyper-parameter setup (e.g., learning rate and weight decay) and 2 num workers IO processes**, running up to 4 GPUs with DataParallel Module.
**What to hand-in:** (1) A report that answers all the questions. The suggested answer format is given below for each question. (2) Question 1 – Question 4.1 requires coding. Please attach your python script and a readme file which lists all the necessary commands to run for each experiment. (3) Question section 4.2 – Question section 7 are essay questions. You only need to provide your answers. If you happen to have code that corroborate your answers, you are welcome to hand in!
**Machines to use:** This lab is designed to be done on NYU HPC. You are welcome to use any GPU cluster that can run 4-GPU job, but do keep the type of GPUs you run consistent.

## Q1: Computational Efficiency w.r.t Batch Size — 5 points

**Problem Description:** Measure how long does it take to complete 1 epoch training using different batch size on single GPU. Start from batch size 32, increase by 4-fold for each measurement (i.e., 32, 128, 512 ...) until single GPU memory cannot hold the batch size. For each run, run 2 epochs, the first epoch is used to warmup CPU/GPU cache; and you should report the training time ( excluding data I/O; but including data movement from CPU to GPU, gradients calculation and weights update) based on the 2nd epoch training.
**Expected Answers:** On Model-X GPU, it takes x1 seconds to train one epoch (w/o data-loading) using mini-batch size 32, x2 seconds using batch-size 128 ... When batch size is larger, it takes longer or shorter time to train, because ...

| | Batch-size 32 per GPU | | Batch-size 128 per GPU | | Batch-size 512 per GPU | |
|---|---|---|---|---|---|---|
| | Time(sec) | Speedup | Time(sec) | Speedup | Time(sec) | Speedup |
| 1-GPU | | 1 | | 1 | | 1 |
| 2-GPU | | | | | | |
| 4-GPU | | | | | | |

Table 1: Speedup Measurement for different Batch Size.

| | Batch-size 32 per GPU | | Batch-size 128 per GPU | | Batch-size 512 per GPU | |
|---|---|---|---|---|---|---|
| | Compute(sec) | Comm(sec) | Compute(sec) | Comm(sec) | Compute(sec) | Comm(sec) |
| 2-GPU | | | | | | |
| 4-GPU | | | | | | |

Table 2: Compute and Communication time for different Batch Size.

## Q2: Speedup Measurement 5 points

**Problem Description:** Measure running time with batch size per GPU you used in Question 1(i.e., 32, 128, ...) on 2 GPUs and 4 GPUs and calculate speedup for each setup. Again, for each setup, run 2 epochs, and only measure the 2nd epoch. When measuring speedup, one should include all the training components (e.g., data loading, cpu-gpu time, compute time)
**Expected Answers:** Table 1 records the training time and speedup for different batch size up to 4 GPUs. Comment on which type of scaling we are measuring: weak-scaling or strong-scaling? Comment on if the other type scaling speedup number will be better or worse than what you we are measuring and give a data-point to corroborate your argument. (hint: you don't have to do any extra experiment or measurement to find this data-point.)

## Q3: Computation vs Communication 15 points

### Q3.1: How much time spent in computation and communication 5 points

**Problem Description:** Report for each batch size per GPU (i.e., 32, 128, 512 ...), how much time spent in computation (including CPU-GPU transferring and calculation) and how much time spent in communication in 2-GPU and 4-GPU case for one epoch. (hint You could use the training time reported in Question 1 to facilitate your calculation)
**Expected Answers:** First, describe how do you get the compute and communication time in each setup. Second, list compute and communication time in Table 2.

### Q3.2: Communication bandwidth utilization 10 points

**Problem Description:** Assume PyTorch DP implements the all-reduce algorithm as described in [3], calculate communication bandwidth utilization for each multi-gpu/batch-size-per-gpu setup.
**Expected Answers:** First, list the formula to calculate how long does it take to finish an allreduce. Second, list the formula to calculate the bandwidth utilization. Third, list the calculated results in Table 3.

## Q4: Large Batch Training 10 points

### Q4.1: Accuracy when using large batch 5 points

**Problem Description:** Report the average training loss and training accuracy for the 5th epoch using the largest batch size per gpu you found in Question 1 with 4 GPUs and compare it with the training loss and training accuracy from Lab 2 (trained with batch size 128).
**Expected Answers:** The average training loss and training accuracy for the 5th epoch for batch size per gpu x on 4 GPUs is x and x.

### Q4.2 How to improve training accuracy when batch size is large 5 points

**Problem Description:** By reading [4], come up with two remedies that can improve training accuracy when batch size is large.
**Expected Answers:** Remedy 1 ... Remedy 2 ...

| | Batch-size-per-GPU 32 | Batch-size-per-GPU 128 | Batch-size-per-GPU 512 |
|---|---|---|---|
| | Bandwidth Utilization(GB/s) | Bandwidth Utilization(GB/s) | Bandwidth Utilization(GB/s) |
| 2-GPU | | | |
| 4-GPU | | | |

Table 3: Communication Bandwidth utilization.

## Q5: Distributed Data Parallel 5 points

**Problem Description:** In order to run across different servers, PyTorch provides Distributed Data Parallel (DDP) [1,5]. One major difference between DP and DDP is that one needs to set up the epoch ID for data loader at the beginning of each epoch training in DDP whereas one doesn't need to specify the epoch ID in DP. By reading DDP document, comment on why one must set the epoch ID in DDP case.
**Expected Answers:** One needs to set up epoch ID because ...

## Q6: What are passed on network? 5 points

**Problem Description:** Are gradients the only message communicated across learners? (*hint: C7 in Lab 2*)
**Expected Answers:** Yes, because ... or No, xxx are also communicated.

## Q7: What if we only communicate gradients? 5 points

**Problem Description:** For the batch size per GPU 512, 4-GPU case, would it be sufficient to communicate only gradients across 4 GPUs? (hint read [4])
**Expected Answers:** Yes, it is okay because ... or No, because ...

# Part-B: Quantization (50 points)

In this lab, you will learn how to generate machine learning models that can be run efficiently on limited hardware resources. We will use the post-training quantization technique, where we reduce the precision of the weights and inputs of a model that has already been trained. Additionally, we will change the data format of our inputs and weights from expensive floating-point numbers to cheap fixed-point numbers. We will use the Google Co-laboratory environment. A template notebook has been prepared which you will edit for this part of the assignment.

Machine learning models are typically trained using 32-bit floating-point data. However, floating-point arithmetic is very expensive (in terms of area, performance, and energy) to implement in hardware. Additionally, 32 bits of precision may be needed during training to capture very small gradient steps, but that much precision is usually unnecessary during inference. Lowering arithmetic precision can save both circuit area and memory bandwidth, helping hardware designers to save costs on several fronts. Therefore ML accelerators typically implement fixed-point arithmetic at much lower precisions. 8-bit unsigned integers are a common target, but some accelerators will go all the way down to single-bit binary arithmetic. There are two broad approaches to quantization: post-training quantization, and quantization-aware training. Quantization-aware training [6] can preserve more accuracy. We explore only post-training quantization in this assignment.

- Go to `https://colab.research.google.com/drive/1QLOlON9NrP2o-oyreWZl5Ur-BuAZqoYb`
- Select *File* → *Save a copy in Drive*
- Select *Runtime* → *Change runtime type*, and then *select GPU* under the Hardware accelerator drop-down menu as shown in Figure 1.

Complete the Colab notebook which contains code that will load the CIFAR10 dataset and train a simple convolutional neural network (CNN) to classify it. You will fill out this notebook (questions and code blocks) to quantize sections of the CNN one-at-a-time.

Submit the copy of the notebook with your answers, plots and comments and a pdf writeup with all your plots and code snippets. Make sure to include your name and student ID in the writeup. For questions that ask for plots, please put the plots in your PDF writeup. For questions that ask you to write code, submit the code sample you wrote in the PDF as well. Make sure your code is nicely formatted and has comments. The grading structure of the notebook is as follows:

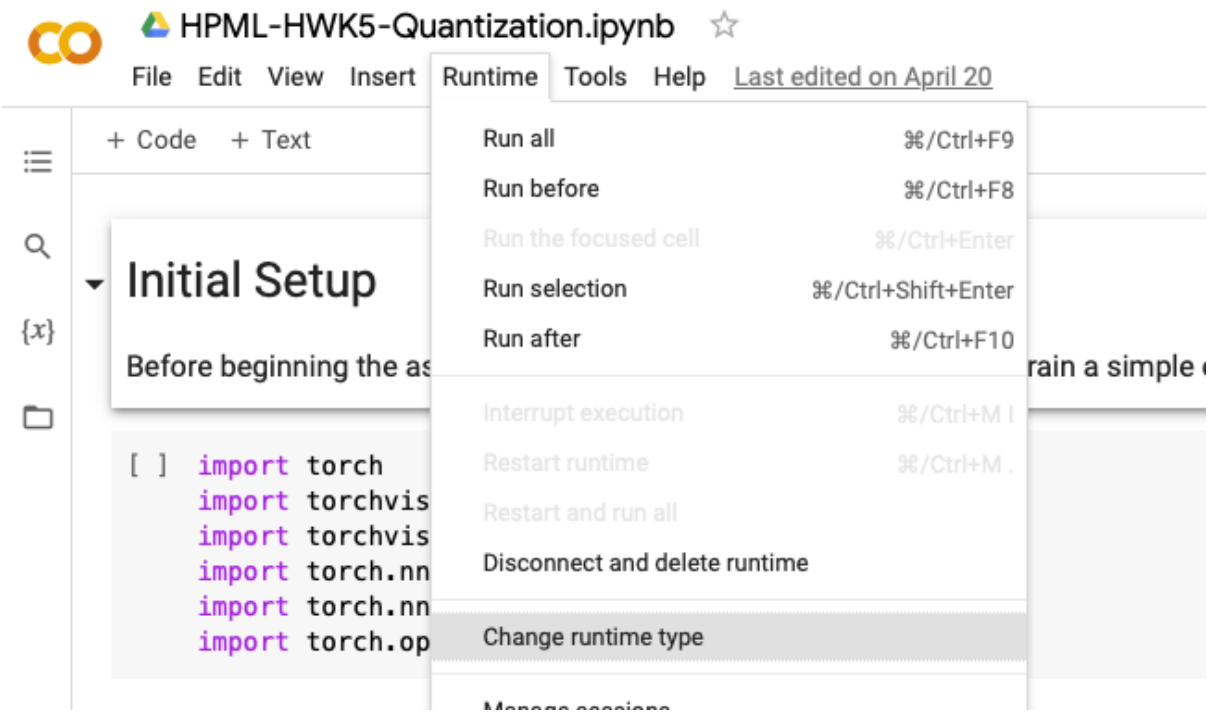


Figure 1: How to switch the Google Colaboratory Runtime

**Q1: Visualize Weights** **10 points**

**Q2: Quantize Weights** **10 points**

**Q3: Visualize Activations** **10 points**

**Q4: Quantize Activations** **10 points**

**Q5: Quantize Biases** **10 points**

# References

[1] PyTorch,*Pytorch Distributed Overview*, Available at https://pytorch.org/tutorials/beginner/dist_overview.html

[2] PyTorch,*Pytorch Data Parallel*, Available at https://pytorch.org/docs/stable/_modules/torch/nn/parallel/data_parallel.html.

[3] P Patarasuk and X Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," J. Parallel Distrib. Comput., vol. 69, pp. 117–124, 2009.

[4] P Goyal, P Dollar, R. B Girshick, P Noordhuis, L Wesolowski, A Kyrola, A Tulloch, Y Jia, and K He, "Accurate, large minibatch SGD: training imagenet in 1 hour," CoRR, vol. abs1706.02677, 2017.

[5] PyTorch, *PyTorch Distributed Data Parallel*, Available at https://pytorch.org/tutorials/intermediate/ddp_tutorial.html.

[6] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, Benoit Jacob et al. https://arxiv.org/abs/1712.05877