DS-UA 301
Advanced Topics in Data Science
Instructor: Parijat Dube
Due: December 15, 2023

Homework 7

General Instructions

This homework must be turned in on both Gradescope and Brightspace by 11:59 pm on the due date. It must be your own work and your own work only—you must not copy anyone's work, or allow anyone to copy yours. This extends to writing code. You may consult with others, but when you write up, you must do so alone. Your homework submission must be written and submitted using Jupyter Notebook (.ipynb). **No handwritten solutions will be accepted**. You should submit:

1. On **Gradescope**: one Jupyter Notebook(.ipynb) containing your solutions for each Problem in this assignment. Each Jupyter Notebook should be named as (*netid*)_hw(*#Homework*)_p(*#Problem*).ipynb.

   For example, *bz2058_hw7_p1.ipynb*.

2. On **Brightspace**: one .zip file containing all the notebooks, datasets, and other supporting materials involved in this homework. The .zip file should be named as (*netid*)_hw(*#Homework*).zip.

   For example, *bz2058_hw7.zip*.

Please make sure your answers are clearly structured in the Jupyter Notebooks:

1. Copy the Template Notebooks provided on Brightspace and write your solutions there.
2. Label each question part clearly. Do not include written answers as code comments. The code used to obtain the answer for each question part should accompany the written answer.
3. All plots should include informative axis labels and legends. All codes should be accompanied by informative comments. All output of the code should be retained.
4. Math formulas can be typesetted in Markdown in the same way as LaTeX. A Markdown Guide is provided on Brightspace for reference.

For more homework-related policies, please refer to the syllabus.

Problem 1 - *Staleness in Asynchronous SGD* 6 points

In a Parameter-Server (PS) based Asynchronous SGD training system, there are two learners. Assume a learner sends gradients to the PS, PS updates weights and a learner pulls the weights from the PS in zero amount of time (i.e. after the learner sends gradients to the PS, it can receive updated weights from PS immediately). Now assume learner 1 runs at about 2.5x the speed of learner 2. Learner 1 calculates gradients $g[L_1, 1]$ at second 1, $g[L_1, 2]$ at second 2, $g[L_1, 3]$ at second 3, $g[L_1, 4]$ at second 4. Learner 2 calculates gradients $g[L_2, 1]$ at second 2.5, $g[L_2, 2]$ at second 5. Updates to weights are instant once a gradient is available. Calculate the staleness (number of weight updates between reading and updating weights) of $g[L_1, 1], g[L_1, 2], g[L_1, 3] g[L_1, 4], g[L_2, 1], g[L_2, 2]$. ($g[L_i, j]$ means i-th learner's j-th calculated gradients).

Problem 2 - *Data Parallelism in Pytorch* 20 points

We are going to experiment with PyTorch's DataParallel Module, which is PyTorch's Synchronous SGD implementation across a number of GPUs on the same server. In particular, we will train ResNet-18 implementation from https://github.com/kuangliu/pytorch-cifar with num_workers=2, running up to 4 GPUs with the DataParallel (DP) Module. Use SGD optimizers with 0.1 as the learning rate, momentum 0.9, and

Homework 7

weight decay 5e-4. For this question, you need to do experiment with multiple GPUs on the same server. You may need to execute this on the NYU Greene Cluster.

Create a PyTorch program with a DataLoader that loads the images and the related labels from the torchvision CIFAR10 dataset. Import the CIFAR10 dataset for the torchvision package, with the following sequence of transformations:

- Random cropping, with size 32x32 and padding 4
- Random horizontal flipping with a probability 0.5
- Normalize each image's RGB channel with mean(0.4914, 0.4822, 0.4465) and variance (0.2023, 0.1994, 0.2010)

The DataLoader for the training set uses a minibatch size of 128 and 3 IO processes (i.e., num_workers=2). The DataLoader for the testing set uses a minibatch size of 100 and 3 IO processes (i.e., num_workers =2). Create a main function that creates the DataLoaders for the training set and the neural network.

1. Measure how long it takes to complete 1 epoch of training using different batch sizes on a 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 warm up the 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. (5)
2. Measure running time with batch size per GPU you used in part 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). (5).
   Expected Answer: Table 1 records the training time and speedup for different batch sizes up to 4 GPUs. Comment on which type of scaling we are measuring: weak-scaling or strong-scaling? Comment on if the other type of scaling was used to speed up the number will be better or worse than what you are measuring.
3. 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). (5)
   Expected Answer: First, describe how you get the compute and communication time in each setup. Second, list compute and communication time in Table 2.
4. Assume PyTorch DP implements the all-reduce algorithm as discussed in the class (reference below), and calculates communication bandwidth utilization for each multi-gpu/batch-size-per-gpu setup. (5)
   Expected Answer: First, list the formula to calculate how long it takes to finish an allreduce. Second, list the formula to calculate the bandwidth utilization. Third, list the calculated results in Table 3.

   References:

   - PyTorch Data Parallel, Available at https://pytorch.org/docs/stable/_modules/ torch/nn/parallel/data_parallel.html.
   - Bringing HPC Techniques to Deep Learning

	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.

	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

Problem 3 - *SSD, ONNX model, Visualization, Inferencing* 24 points

In this problem, we will be inferencing the SSD ONNX model using the ONNX Runtime Server. You will follow the GitHub repo and ONNX tutorials (links provided below). You will start with a pre-trained Pytorch SSD model and retrain it for your target categories. Then you will convert this Pytorch model to ONNX and deploy it on the ONNX runtime server for inferencing.

1. Download pretrained pytorch MobilenetV1 SSD and test it locally using Pascal VOC 2007 dataset. Show the test accuracy for the 20 classes. (2)

2. Select any two related categories from Google Open Images dataset and finetune the pretrained SSD model. Examples include Aircraft and Aeroplane, Handguns and Shotguns. You can use `open_images_downloader.py` script provided on GitHub to download the data. For finetuning you can use the same parameters as in the tutorial below. Compute the accuracy of the test data for these categories before and after finetuning. (3+3)

3. Export the Pytorch model to ONNX using `torch.onnx.export()` function and save it. When you export the model the function will execute the model, recording a trace of what operators are used to compute the outputs. Because export runs the model, we need to provide an input tensor `x`. The values in this can be random as long as it is the type and size. Use a dummy random tensor. (3)

4. Load the saved model using `onnx.load` and verify the model's structure using `onnx.checker.check_model`. (3)

5. Next run the model with ONNX Runtime (ORT). You first need to create an inference session for the model and then evaluate the model using the `run()` API. (3)

6. Does the output of PyTorch (from `torch.out`) and ONNX Runtime match? What precision did you use to match? (2)

7. Test the inferencing set-up using 1 image from each of the two selected categories. For this, you will need to load the images, preprocess them, and then do inference using the `run()` API from ORT. (5)

8. **[BONUS]** Parse the response message from the ORT and annotate the two images. Show inferencing output (bounding boxes with labels) for the two images. (5)

For parts 1 and 2 refer to the steps in the Github repo in the references. For parts 3-6 you can refer to the Pytorch tutorial in the reference and the associated notebook. For part 7 you can refer to Pytorch tutorial for creating an inference request to ORT and to ONNX tutorial on how to preprocess an image. For part 8 you need to adopt the code at ONNX tutorial on parsing the ORT response.

References

- Github repo. Shot MultiBox Detector Implementation in Pytorch.
  Available at https://github.com/qfgaohao/pytorch-ssd

- Pytorch tutorial. Exporting a model from Pytorch to Onnx and running it using Onne runtime.
  Available at https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

- ONNX tutorial. Inferencing SSD ONNX model using ONNX Runtime Server.
  Available at https://github.com/onnx/tutorials/blob/master/tutorials/OnnxRuntimeServerSSDModel.ipynb

- Google. Open Images Dataset V5 + Extensions.
  Available at https://storage.googleapis.com/openimages/web/index.html

- The PASCAL Visual Object Classes Challenge 2007.
  Available at http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

Problem 4 - *Deep Reinforcement Learning* 20 points

This question is based on Deep RL concepts discussed in the lecture. You need to refer to the papers by Mnih et al., Nair et al., and Horgan et al. to answer this question. All papers are linked below.

1. Explain the difference between episodic and continuous tasks. Give an example of each. (2)

2. What do the terms exploration and exploitation mean in RL? Why do the actors employ ϵ-greedy policy for selecting actions at each step? Should ϵ remain fixed or follow a schedule during Deep RL training? How does the value of ϵ help balance exploration and exploitation during training? (1+1+1+1)

3. How is the Deep Q-Learning algorithm different from Q-learning? You will follow the steps of the Deep Q-Learning algorithm in Mnih et al. (2013) page 5, and explain each step in your own words. (3)

4. What is the benefit of having a target Q-network? (3)

5. How does experience replay help in efficient Q-learning? (2)

6. What is prioritized experience replay? Explain how priority is calculated for different experiences. (3)

7. Compare and contrast GORILA (General Reinforcement Learning Architecture) and Ape-X architecture. Provide three similarities and three differences. (3)

References

- Mnih et al. Playing Atari with Deep Reinforcement Learning. 2013
  Available at https://arxiv.org/pdf/1312.5602.pdf
- Nair et al. Massively Parallel Methods for Deep Reinforcement Learning. 2015
  Available at https://arxiv.org/pdf/1507.04296.pdf
- Horgan et al. Distributed Prioritized Experience Replay. 2018
  Available at https://arxiv.org/pdf/1803.00933.pdf

Problem 5 - *ML Cloud Platforms* 20 points [BONUS]

In this question, you will analyze different ML cloud platforms and compare their service offerings. In particular, you will consider ML cloud offerings from IBM, Google, Microsoft, and Amazon and compare them on the basis of the following criteria:

1. Frameworks: DL framework(s) supported and their version. (4)
   Here we are referring to machine learning platforms that have their own inbuilt images for different frameworks.
2. Compute units: type(s) of compute units offered, i.e., GPU types. (1)
3. Model lifecycle management: tools supported to manage ML model lifecycle. (2)
4. Monitoring: availability of application logs and resource (GPU, CPU, memory) usage monitoring data to the user. (3)
5. Visualization during training: performance metrics like accuracy and throughput (2)
6. Training job description: training job description file format. Show how the same training job is specified in different ML platforms using an example job. Identify similar fields in the training job file for the 4 ML platforms through an example. (8)