Lab 4 Report

ECE-GY 9143 Intro to High Performance Machine Learning

Aditya Wagh (amw9425)

April 17, 2022

Question 1: Computational Efficiency w.r.t Batch Size

On an RTX8000 GPU,

- it takes 14.9203 seconds to train one epoch (w/o data-loading) using mini-batch size 32,
- 3.8875 seconds using batch-size 128
- 1.1585 seconds using batch-size 512
- 0.5569 seconds using batch-size 2048
- 0.2947 seconds using batch-size 8192

Beyond a batch size of 8192, the model wouldn't fit on the RTX8000 GPU. When batch size is larger, it takes a shorter time to train, which reflects the lower overhead associated with loading a smaller number of large batches, as opposed to many small batches sequentially.

Question 2 Speedup Measurement

	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	16.0478	1	4.3415	1	1.5882	1
2-GPU	13.0978	1.22	3.6809	1.17	1.5409	1.0306
4-GPU	9.0060	1.78	2.7344	1.58	2.7108	0.5858

	Batch-size 2048 per GPU		Batch-size 8192 per GPU	
	Time(sec)	Speedup	Time (sec)	Speedup
1-GPU	1.3481	1	2.7618	1
2-GPU	1.8382	0.73	4.9845	0.55
4-GPU	2.9051	0.46	10.7934	0.25

Table 1: Speedup Measurement for different batch sizes.

Training many state-of-the-art deep neural networks is a very compute-intensive task. It can take hours, days or even weeks to train a model with a single computational device, such as a CPU or GPU. To speed up the training, you have to scale out, to distribute the computations to multiple devices. The most commonly used approach to distributing the training is data parallelism when every computational device possesses its own replica of a model and computes a model update based on its own shard of data. Two options are possible for data parallelism: Strong and weak scaling.

Strong Scaling

Strong scaling assumes that the problem size remains the same and we vary the number of computational devices. From a deep learning point of view, it means that we fix a batch size and vary the number of CPUs/GPUs to train a neural network. With strong scaling, the effective batch size is constant, and the per-device batch size is varying. In the case of synchronous training, strong scaling with multiple devices is equivalent to training with a single device. The effective batch will be the same, and model updates will happen based on gradients computed for the same number of training samples.

Weak scaling

Weak scaling assumes we keep the amount of work per computing device per iteration fixed. With N compute devices we end up solving N times larger problems than with one. In the deep learning world, it means we keep per-device batch size fixed.

The speedup of weak scaling is calculated in the following way: speedup = $\frac{t_1}{t_n}$

Where t_1 is the time of solving a problem with one compute device and t_N is the time to solve the N times the larger problem with N compute devices. Ideally, the t_N time is exactly the same as t_1 and we get ideal efficiency of 100%. With weak scaling, the per-device batch size is constant and the effective batch size increases with the assignment of more compute devices to the training job. It is much easier to utilize a larger number of computing devices efficiently with weak scaling, as the amount of work per unit doesn't decrease when more units are added. At the same time, weak scaling may lead to very large effective batches, when convergence will suffer, and it won't be faster after all.

As per the above discussion, the scaling in our case is weak scaling. We are keeping the batch size per GPU constant while increasing the number of GPUs.

Question 3: Computation vs Communication

Time spent in computation and communication

The time for an all-reduce operation is defined as the time when all processes start the operation until the last process receives the last message (and computes the final result). The time spent in computation is equal to the training time we calculate in the lab4.py script. It includes the time to load data, time to move the data to GPU, and time to compute loss, gradients and perform backpropagation.

In a general-purpose all-reduce operation, data items are independent of each other. The reduction on different items is independent of one another. Hence, the amount of data that must be communicated in order to complete an all-reduce operation on X items is equal to X times the amount for the single-item all-reduce operation.

The time required for communication will be

$$t_{comm} = t_N - \frac{t_1}{N}$$

	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	13.0978	5.0739	3.6809	1.5105	1.5409	0.7468
4-GPU	9.0060	4.9940	2.7344	1.6490	2.7108	2.3137

	Batch-size 2048 per GPU		Batch-size 8192 per GPU	
	Compute (sec)	Comm(sec)	Compute(sec)	Comm(sec)
2-GPU	1.8382	1.1641	4.9845	3.6036
4-GPU	2.9051	2.5680	10.7934	10.1029

Table 2: Computation and communication time for different batch sizes.

Communication bandwidth utilization

The time taken to perform the Allreduce operation on an array of size S is

$$T = M\frac{2(N-1)}{BN}$$

where M is the message size, B is the bandwidth and N is the number of workers.

After some rearranging, the bandwidth required for the operation is

$$B = M\frac{2(N-1)}{TN}$$

The message size for the operation would be

$$\frac{4 \times \#\text{params} \times (\#\text{training samples})}{\text{batchsize} \times \#\text{workers}} = \frac{4 \times 11,173,962 \times 50,000}{\text{batchsize} \times \#\text{workers}} \text{ bytes} \approx \frac{2081}{\text{batchsize} \times \#\text{workers}} \; GB$$

and T is t_{comm}. Using the above results, we get the bandwidth as mentioned in Table 3.

	Batch-size-per-GPU 32	Batch-size-per-GPU 128	Batch-size-per-GPU 512
	Bandwidth Utilization (GB/sec)	Bandwidth Utilization (GB/sec)	Bandwidth Utilization(GB/s)
2-GPU	6.4084	5.3816	1.3454
4-GPU	6.5109	4.9296	0.8783

	Batch-size-per-GPU 2048	Batch-size-per-GPU 8192
	Bandwidth Utilization (GB/sec)	Bandwidth Utilization (GB/sec)
2-GPU	0.4364	0.0352
4-GPU	0.1978	0.0126

Table 3: Communication Bandwidth Utilization

Question 4 Large batch training

Accuracy when using large batch size

Lab 4: The average training loss and training accuracy for the 5th epoch for batch size per GPU 8192 on 4 GPUs is 2.878 and 12.640.
Lab 2: The average training loss and training accuracy for the 5th epoch for batch size per GPU 128 on 1 GPU is 0.847 and 70.098.

Remedies for improving training accuracy when batch size is large.

According to the paper **Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour** Goyal et al, there are two remedies for improving training accuracy when using large batch sizes:

- **Warmup.** *Constant Warmup:* The main idea is to start with a low learning rate to solve rapid changes in the initial network. A sudden change in the learning rate causes the training error to spike. *Gradual Warmup:* Ramp up the learning rate from a small to a large value. Start from a learning rate of η and increment it by a constant amount at each iteration such that it reaches $\hat{\eta} = k\eta$ after 5 epochs

- **Linear Scaling of Learning Rate** When the mini-batch size is multiplied by k, multiply the learning rate by k. Eq (1) represents the weight update equation for the η learning rate. Eq (2) represents the weight update equation for $k\eta$ learning rate.

$$w_{t+k} = w_t - \eta\frac{1}{n}\sum_{j<k}\sum_{x\in\mathcal{B}_j}\nabla l\left(x, w_{t+j}\right) \tag{1}$$

$$\hat{w}_{t+1} = w_t - \hat{\eta}\frac{1}{kn}\sum_{j<k}\sum_{x\in\mathcal{B}_j}\nabla l\left(x, w_t\right) \tag{2}$$

Assuming that the gradients are equal, the only way for these equations to be the same is if we set the second learning rate to k times the first learning rate.

Question 5 Distributed data parallel

It's for determinism across runs, i.e. same samples are fetched to workers across multiple runs even across distributed training.

Question 6 Comments on messages communicated across learners

No, gradients aren't the only things that are communicated between learners. They will communicate all the learnable parameters. Since the model has batch normalization layers, the beta and gamma values of each layer would also be communicated.

Question 7 Comments on only communicating gradients

The BN statistics should not be computed across all workers, not only for the sake of reducing communication, but also for maintaining the same underlying loss function being optimized.