CORNELL TECH | HOME OF THE JACOBS TECHNION-CORNELL INSTITUTE

[Fall 2025] ECE 5290/7290 and ORIE 5290
Distributed Optimization for Machine Learning and AI
Homework 4

Gradescope Due: November 14th at 5PM

Objective of This Assignment

The objective of this assignment is to gain a foundational understanding of the key concepts, practical implementation of distributed learning, and how it can accelerate the training. You will begin by learning the crucial concepts and implementing your first distributed program to get your feet wet. Based on that, you will implement one of the central operations in distributed learning `all-reduce` and compare the complexity of different implementations. Finally, you will be guided to implement your distributed training algorithm.

We will provide a starter code framework on the CANVAS course website in the format of Python notebooks. There are two kinds of questions:

- Q1 Programming: Even though the framework is offered, key functions in the notebook are left blank with a placeholder `raise NotImplementedError`. You are responsible for completing the functions. Remove the placeholder after you complete them.
- Q2 Question: You are responsible for answering the questions posed in the notebook.

Note: There is no autograder for this project, since you can verify that your code outputs the correct thing by just comparing it to the baseline methods provided. Grading will be performed based on your lab report and on manual inspection of your code. Furthermore, the experiments in this project can take some time to complete. On my computer, they take about 15 minutes to run in total (across all experiments). Please ensure you leave enough time to complete the experimental exploration.

Task 0: Setup the environment (0%)

The first task will guide you to get your environment ready. In this homework, only `PyTorch` package is required. If you have already had a proper environment, you could skip this part. **Note: Do not use the package except PyTorch.**

The starter code is provided in the format of `Python notebook (ipynb)`. You could run the notebook on various platforms, like `Jupyter Notebook`, `Google Colab`, and `VS Code`. We suggest using `Conda` package manager to set up the environment. Run `conda env create -f environment-cpu.yml` to create a virtual environment and use `conda activate ECE5290-HW3-cpu` to activate it.

Task 1: Distributed operations (15 points)

You will utilize the `PyTorch Distributed` library, which comprises a collection of parallelism modules, a communication layer, and infrastructure for launching and debugging large-scale training jobs. The library

can use either a CPU or GPU backend, depending on your hardware and configuration. If you have a server with multiple GPUs, you could use `NCCL` as the backend; otherwise, you could use `Gloo` as the backend. In the homework, only the CPU is required. Refer to the official PyTorch Distributed Documentation for guidance.

Step 1: Distributed "Hello World" (5 points)

The first step of Task 1 is to get familiar with the concept of distributed computing. Additionally, you need to let each worker say hello to you, which ensures you have a correct distributed environment.

In distributed computing, two fundamental concepts are `world size` and `rank`, which provide the necessary context for managing data, synchronizing model updates, and orchestrating the entire distributed training workflow. `World size` refers to the total number of processes participating in a distributed computing job. This is the global count of all workers, which could be across multiple machines or multiple GPUs on a single machine. `Rank`, on the other hand, is the unique identifier assigned to each individual process within the distributed group. Each process has a distinct rank ranging from 0 to `world size` - 1. This unique identifier is crucial for coordinating communication and ensuring that each process knows its specific role in the training task, such as which partition of data or which subset of a model it is responsible for.

The entry of a distributed program is something like `spawn(func)`, where `spawn` functions generate a series of workers and `func`. In this task, you will define different `func` according to different requirements. As the first step, you need to define a `func` to let workers say hello to you.

Requirements: Let each worker print its `rank` and your name. For example, if you are Alice, your output should be like

```
Rank 1: Hello, Alice!
Rank 2: Hello, Alice!
Rank 3: Hello, Alice!
Rank 4: Hello, Alice!
```

Step 2: Communication among workers (10 points)

In the second step, you will get familiar with the primitives `send` and `recv` functions, which enable point-to-point communication between two workers.

Requirements: You will need to read their documents and answer some questions posed in the notebook. Furthermore, you need to implement a code that allows workers to send messages to others.

Task 2: All-reduction operation (50 points)

In this task, you will implement a key collective communication operation, `all-reduce`, using only the primitive `send` and `recv` functions. Your implementation should not rely on any other collective communication functions. Let x_n be a tensor on worker n, the result of `all-reduce` is that all workers have a copy of $\sum_{n=0}^{N-1} x_n$. As we will see in Task 3, `distributed SGD` involves `all-reduce` at each iteration. Therefore, `all-reduce` plays a critical role in the training. You will compare the efficiency of two major methods to implement `all-reduce`.

Step 1: Tree-all-reduce (20 points)

In the first step, you will be guided to implement the `Tree-all-reduce` algorithm, which operates in two distinct phases to achieve a global reduction across all processes. In each phase, the data flow follows a binary tree communication pattern.

Phase I: Reduce-to-Root. Phase I aims to reduce the data to one specific worker. In this phase, data flows from the leaves of the tree up to the root. In the example in Figure 1, the first row shows the initial state, with four independent workers (P1, P2, P3, P4) at a leaf node that start with their local data x_n. In each communication step, a worker n sends its current data x_n to its parent node n'. The parent node receives data from its children, performs the reduction operation (e.g., summation), and combines the received data with its own local value, i.e., $x_{n'} \leftarrow x_{n'} + x_n$. In the example, the leaves of the tree (P2 and P4) send their data to their respective parents (P1 and P3). P1 receives data from P2, and P3 receives data from P4. This process repeats, with intermediate nodes sending their partially reduced values further up the tree, until the root node has received and processed data from all its children, at which point it holds the final, globally reduced value. In our example, the root worker, P1, holds the final, globally reduced result from all four processes.

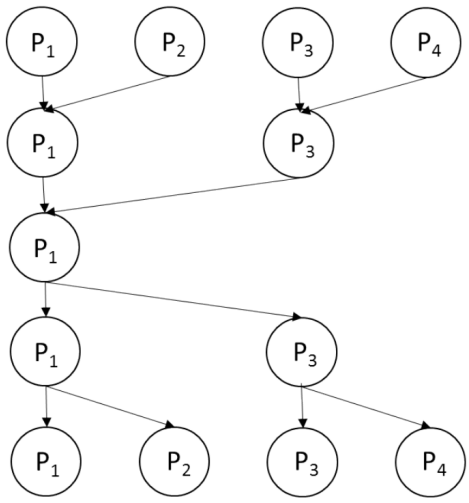

Figure 1: Data flow of Tree-all-reduce for four processes (P1, P2, P3, P4) across two distinct phases.

Phase II: Broadcast-from-Root: The second phase ensures all workers have a copy of the final reduced result. In this phase, the final reduced value, now residing at the root (P1), is broadcast down the tree to all other workers. The root sends the result to its children. Each child, upon receiving the value, then sends it to its own children, and so on. This process continues until the result is disseminated to every worker.

Requirements: Answer the following questions or finish the functions in the notebook:

(a) (5 points) Given a system with N workers, how many communication steps are needed to complete `Tree-all-reduce`?

(b) (5 points) At the i-th communication step in Phase I, worker n sends a message to worker n'. Write n' as a function of i and n.

(c) (10 points) Implement `Tree-all-reduce` in the notebook.

Step 2: Ring-all-reduce (20 points)

The `Ring-all-Reduce` is a two-phase process, where the data flow follows a ring topology, i.e., each worker sends data to its neighbor and receives data from its other neighbor.

Phase I: Scatter-reduce. Similar to `Tree-all-Reduce`, Phase I of `Ring-all-Reduce` aims to reduce the data. However, `Ring-all-Reduce` divides the data x_n into N chunks (assume the size of x_n is divisible by N), where N is the number of workers (i.e., `world_size`). The goal of this phase is for each worker to end up with the final reduced data for one specific chunk. Figure 2 illustrates the process of Phase I in a

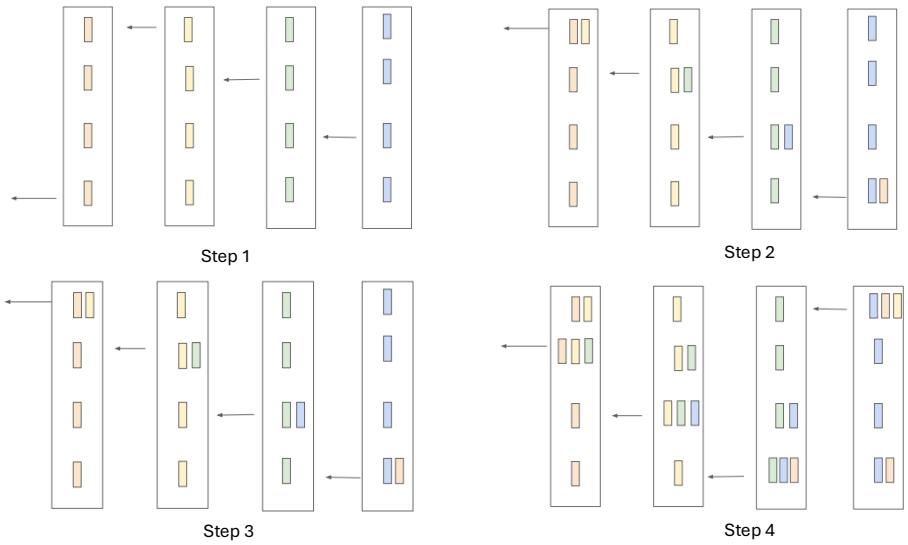

Figure 2: Phase I of Ring-all-reduce: Scatter-Reduce.

system with 4 workers. At the first communication step, worker n sends its $((n-1) \bmod N)$-th chunk to its left neighbor $(n-1) \bmod N$ and receives a chunk from its right neighbor $(n+1) \bmod N$. The received chunk is then added to the corresponding local chunk. At the second step, worker n sends its next chunk to its left neighbor. This send-receive-reduce process is repeated $N-1$ times. Ultimately, each worker has accumulated the sum of one specific chunk from all other processes, and that chunk's value is now its final, reduced value.

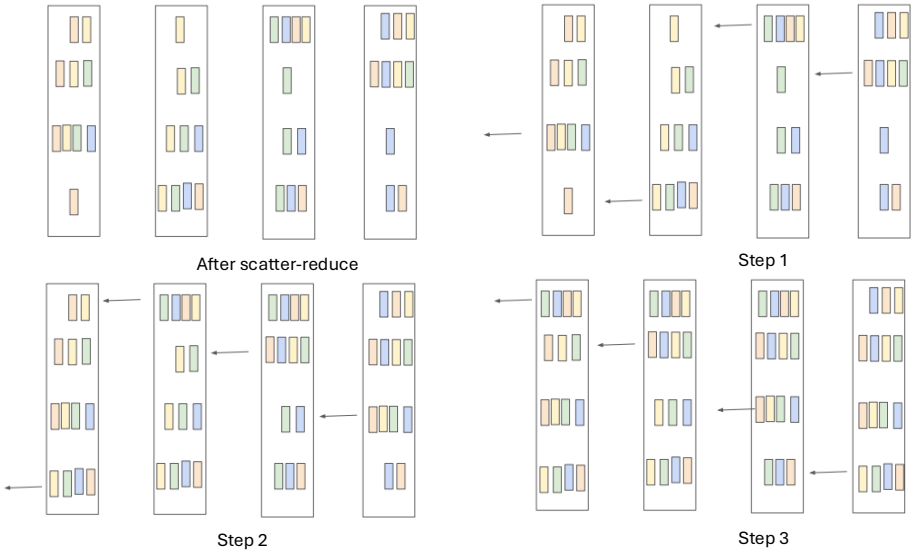

Figure 3: Phase II of Ring-all-reduce: All-Gather.

Phase II: All-Gather. As the upper-left sub-figure in Figure 3 shows, the n-th worker has its $(n+2) \bmod N$ chunk reduced after Phase I. The goal of Phase II is for each worker to share their final, reduced chunk with all the other workers, so that every worker has a complete copy of the final, reduced data. The communication pattern in Phase II is the same as in Phase I. Each worker sends its newly finalized chunk (the one it reduced

in Phase I) to its left neighbor and receives a chunk from its right neighbor. The received chunk is placed into the correct position in the local tensor. This process is repeated $N-1$ times, with each worker forwarding the chunks it receives around the ring. Ultimately, every worker has received and collected all the other $N-1$ reduced chunks.

Requirements: Answer the following questions or finish the functions in the notebook:

(a) (5 points) At the i-th step, the c-th data chunk at the n-th worker is sent to its left neighbor. Write c as a function of i and n.

(b) (15 points) Implement `Ring-all-reduce` in the notebook.

Step 3: Comparison of complexity (10 points)

Let the latency in the communication between two workers be t; and the bandwidth be b; the data size be D. Consider a distributed system with N workers. Answer the following questions in the notebook:

(a) (2 points) How much time does it take to send data of size D from one worker to another?

(b) (2 points) How much time does it take to complete a `Tree-all-reduce`?

(c) (2 points) How much time does it take to complete a `Ring-all-reduce`?

(d) (4 points) Discuss when we should use `Tree-all-reduce` and when we should use `Ring-all-reduce`.

Task 3: Distributed training (35 points)

In this task, you will combine the concepts from the previous tasks. After performing a forward pass and backward pass on the LeNet-5 model (see Homework 3) in each worker, you need to aggregate the gradients across all workers. You are allowed to use `all-reduce` function provided by `Pytorch Distributed` library.

Algorithm 1 Distributed SGD

Require: step size α; initialization x^0
1: **for all** $k = 0, 1, 2, \cdots, K$ **do**
2: **for all** workers $n = 1, 2, \cdots, N$ **do**
3: Compute stochastic gradient G_n^k using local dataset
4: Compute $x_n^{k+\frac{1}{2}} = x^k - \alpha G_n^k$
5: Run `all-reduce` sub-program and divide the result by N, yielding $x^{k+1} = \frac{1}{N}\sum_{n=1}^{N} x_n^{k+\frac{1}{2}}$
6: **end for**
7: **end for**

Step 1: Distributed SGD (20 points)

Algorithm 1 outlines a process for `Distributed Stochastic Gradient Descent (Distributed SGD)` designed to train a model across multiple workers. The method involves two main loops: an outer loop iterating through training iterations $k = 1, 2, \cdots, K$, and an inner loop where all workers ($n = 1, 2, \ldots, N$) participate in parallel. In each training iteration, every worker independently computes a stochastic gradient G_n^k using its own local dataset. Following this, each worker updates its local model from x^k to $x_n^{k+\frac{1}{2}}$ using the step size

α and its local gradient. The key to the distributed approach lies in the subsequent step, where all workers collectively run an `all-reduce` sub-program. This operation aggregates the updated parameters from all N workers, computes their average, and distributes the final averaged parameters x^{k+1} back to every worker. This synchronization ensures that all workers are aligned with a consistent model state before proceeding to the next training step.

Requirements: Read and understand the provided framework and complete the rest.

Step 2: Validating the speedup (15 points)

Requirements: Answer the following questions.

(a) (10 points) Let $k^*(N)$ be the iterations that `Distributed SGD` with N workers reaches the 97% accuracy on test set. The speedup of the system with N workers is defined as $k^*(1)/k^*(N)$. Set the `world-size` as 1, 2, 4, 8, and run `Distributed SGD`. Compute the speedup.

(b) (5 points) Ideally, a well-scalable distributed system enables the speedup to increase linearly as more devices are deployed, i.e., $k^*(1)/k^*(N) = N$. Does `Distributed SGD` achieve linear speedup? If not, speculate what the bottleneck is.