19.05.2025 Week 13 exercises: Distributed And Decentralized Machine Learning

Exercise 1 (Comparing Network Requirements in Federated Learning (FL) with Decentralized Learning (DL))

You run a FL server at home using the best Internet plan of your Internet Service Provider (ISP), which provides 1 Gbit/s download speed and 100 Mbit/s upload speed (see 2021 offerings by UPC as an example). Further suppose that:

- you are training a Deep Convolutional Network, such as LeNet [1], with 85098 parameters, with each parameter requiring a floating point value (32-bits) in memory; network latency is negligible;
- that the averaging time on the server is negligible;
- that the network, the server, and all clients are perfectly reliable and of similar performance;
- your ISP can fully and continuously guarantee the advertised download and upload speeds;
- your Federated Learning algorithm works in a synchronous fashion, i.e. all clients download the latest model parameters, then all clients update their parameters with a local gradient step, and then all clients upload the updated version of their model;
- performing a local gradient step takes 60 seconds, with no variance;
- that downloading or uploading the model parameters should take no more than 60 seconds (otherwise it would dominate over the cost of computation).

What is the maximum number of participants n_{max} you could support concurrently without exceeding the 60 seconds limit for either downloading or uploading parameters?

$$
\begin{aligned}
t_{\text{download}} = \quad & n_{\text{download}} * 85098 \text{ params} * 32\frac{bits}{param} * \frac{1}{10^9 \frac{bits}{s}} \leq 60s \\
n_{\text{download}} \leq \quad & \frac{60s * 10^9 \frac{bits}{s}}{85098 \text{ params} * 32 \frac{bits}{params}} \approx 22033 \text{ participants} \\
n_{\text{upload}} \leq \quad & \frac{60s * 10^8 \frac{bits}{s}}{85098 \text{ params} * 32 \frac{bits}{params}} \approx 2203 \text{ participants} \\
n_{\text{max}} = \quad & min(n_{\text{download}}, n_{\text{upload}}) = 2203 \text{ participants}
\end{aligned}
$$

Exercise 2 (Economics of FL)

You are an Internet startup training the word prediction model of Google, in their Federated Learning paper (Section 3 "Large-scale LSTM experiments" in [3]) on a Cloud Provider. Compute the savings you could make in download costs if you had trained with DL instead.

For FL, assume the following:

1. parameters are represented as 32-bit floating point values
2. there are a total of 4,950,544 parameters
3. 200 clients participate in each round
4. training lasts 1000 rounds
5. you do not pay to send the models from participating clients to your Cloud Provider (ex: Amazon Cloud Pricing)
6. you pay $\frac{0.03\$}{2^{30}B}$ to download the updated model from the Cloud Provider to participating clients (ex: Amazon Cloud Pricing)
7. computation costs on the Cloud Provider are negligible.

For DL, assume you are not paying for bandwidth beyond your regular ISP subscription and you are using a local machine that you already own for computation.

$$
\begin{aligned}
b = & \quad 4,950,544 \ \tfrac{\text{parameters}}{client} * 4 \tfrac{bytes(B)}{\text{parameter}} = 19,802,176 \tfrac{B}{client} \\
C_{FL} = & \quad n_{\text{rounds}} * n_{\text{clients}} * b * \tfrac{0.03\$}{2^{30}B} \\
C_{FL} = & \quad 1000 \text{ rounds} * 200 \ \tfrac{\text{clients}}{\text{round}} * 19,802,176 \tfrac{B}{client} * \tfrac{0.03\$}{2^{30}B} \approx 110.65\$
\end{aligned}
$$

You could have saved roughly 111$ in download costs.

Exercise 3 (Parameter server v/s All-reduce)

1. Describe how parameter server and all-reduce are two implementations of the same algorithm.
   At the start of each training round, in the absence of failures, each worker will have the parameters. Moreover, the gradient step that an individual worker takes is the same in both cases. The only difference resides in how the communication and aggregation occurs. Therefore, the final output in both cases will be the same, and hence, the two cases are implementations of the same algorithm.
2. In the parameter server architecture, what is the bottleneck in terms of communication? What is the bottleneck in terms of computation?
   The bottleneck in terms of communication is the parameter server, which has to receive all the gradients from all the workers, and send back the updated parameters to all the workers. The bottlenecks in terms of computation are the workers, that have to compute

the gradients for all the parameters.

3. In the all-reduce architecture, what is the bottleneck in terms of communication? What is the bottleneck in terms of computation?
   The bottleneck in terms of communication is distributed from one node (in the case of parameter server) to the entire network, where the aggregation now happens in multiple steps. The bottleneck in terms of computation are still the workers, that have to compute the gradients for all the parameters.

4. Which architecture is more robust to failures? Why?
   The all-reduce architecture is more robust to failures, because if one worker fails, the other workers can still communicate with each other and compute the gradients for all the parameters. In the parameter server architecture, if the parameter server fails, the workers cannot communicate with each other anymore, and cannot compute the gradients for all the parameters. The computation of the worker that failed will naturally be excluded from the round.

Exercise 4 (FL/DL aggregation)

Distributed learning systems rely on the aggregation of data produced by several computing nodes in order to iteratively produce improved versions of a machine learning model.

In the **federated learning** setting, a central server aggregates (averages) the local updates from the edge nodes (e.g., mobile phones) before sending the aggregated model back to a new set of nodes that will work on the next training iteration (i.e., all of them will train on the same new model aggregated by the server).

In the **decentralized learning** setting, on the other hand, there is no central aggregator. In order to compute aggregated models, nodes can only rely on updates shared by their neighbors. Thus, different nodes may train upon distinct models in a given training round. Before the next training iteration, this locally-trained model will be averaged with the updates received from neighbor.

Assume that the goal of a given distributed learning is to collectively achieve a global model that solves an image classification task using stochastic gradient descent (SGD) for optimizing the objective function F, i.e.,

$$\mathbf{x}_i^{(t;k)} \leftarrow \mathbf{x}_i^{(t;k-1)} - \eta \nabla F\big(\mathbf{x}_i^{(t;k-1)}; \mathbf{b}\big)$$

where $\mathbf{x}_i$ are the model parameters on node i, t is the epoch, k is the local step, η is the learning rate and $\mathbf{b}$ is a mini-batch of training data.

Typically, the gradients ∇F are shared in the context of federated learning, whereas the parameters $\mathbf{x}_i$ are shared in the context of decentralized learning. Why? Briefly describe how the choice between gradients or parameters is impacted by the distributed learning architecture, namely: (1) federated learning and (2) decentralized learning.

In Federated Learning, at the start of each training round, the central server sends the current model to all the workers. Therefore, each worker has the same model at the start of each training round. The workers can send back the updates in both forms, either in the form of gradients (parameter change), or the actual parameters. For implementation, and making

optimizations easy, usually, gradients are preferred in this case because they are centered around zero and are easily compressible.

On the other hard, in Decentralized Learning over an overlay topology, the workers do not have the same model at the start of each training round because of the absence of all-reduce. Therefore, gradients of one worker have no meaning for another worker. Therefore, the workers must send the actual parameters to each other

Exercise 5 (From Gossip to Learning)

You have seen how gossip can be used to compute the average of the attributes held by nodes in a network. Decentralized learning is doing something similar, but with a twist: instead of just averaging attributes, we are iteratively training and averaging attributes. **Can you modify the gossip algorithm that computes the average of the attributes to instead perform a variant of decentralized learning?** Clearly explain what the active and passive threads would do in your algorithm. Pinpoint the bottleneck(s) of your algorithm.

```
1: loop                                                    ▷ Active Thread
2:     wait Δ
3:     x ← SGD(x)
4:     p ← Random Peer
5:     send(p, x)
6:
7: procedure OnReceive(m)                                  ▷ Passive Thread
8:         aggregate(m.x)
9: end procedure
10:
11: procedure aggregate(m.x)
12:         x ← (m.x + x)/2
13: end procedure
```

You can read more about the algorithm in the original paper [4]. Bottleneck is the waiting time Δ and the time for SGD. Additionally, since the peer selection is random, one node may be selected by multiple other nodes to send the model to in a round. This can cause the bandwidth to be unbalanced across nodes in a round.

Exercise 6 (Decentralized Learning)

Decentralized parallel stochastic gradient descent (D-PSGD) [2] is an optimization algorithm often used for training models in DL. In each iteration of D-PSGD a node performs one or multiple steps of mini-batch SGD on their local model using their local data. Following this, the node transmits their local model to all direct neighbors in the graph. Upon receiving all models from all their neighbors a node aggregates the received models and their own into a new model that is used as their starting local model for the next iteration of D-PSGD.

There is a system containing 128 nodes organized in a graph with an average degree of 8. Assume that all nodes and connections between them are reliable and there are no stragglers. Furthermore, the distribution of each local dataset follows the distribution of the global dataset

(IID data). The nodes perform 50 steps of D-PSGD, followed by one all-reduce round (assume naive all-reduce without any optimizations). What is the total number of messages exchanged in the given system during this training?

You have a global test set to evaluate each of the local models at different points during the described training process and measure their test loss. Take two arbitrary nodes from the graph and imagine how their respective test loss curves would look like during the training. What are some general conclusions you can come from from this imagination?

At each step of D-PSGD, a node sends its own model to each of their neighbors, so in other words, the number of messages is equal to the total degree of the graph these nodes are forming: **128 * 8**.

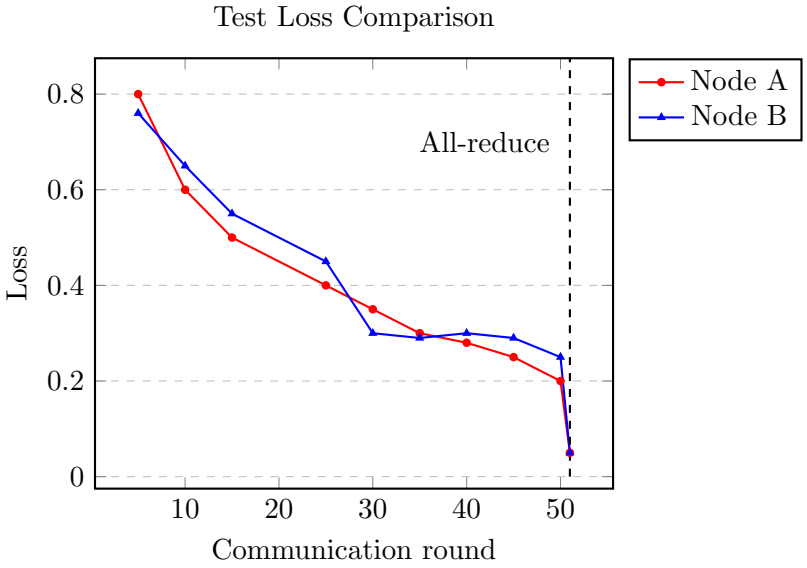
There are 50 D-PSGD, so the total number of message during them is **50 * 128 * 8**.

Naive All-reduce is basically D-PSGD on a complete graph: every node sends their model to every other node. As we've seen the number of messages exchanged in D-PSGD is equal to the total degree of a graph, which is in this case: **128 * 127**.

Hence, the total number of exchanged messages is the sum of number of messages exchanged in D-PSGD and during All-reduce:

$$50 * 128 * 8 + 128 * 127 = 67456.$$

An example of possible training curves:



General takeaways:

- If all nodes use the same model architecture, we expect them to show similar convergence, because the data is distributed in IID fashion (i.e. the distribution of the data at each node is similar).
- The test loss during D-PSGD isn't exactly the same even though we test on the same test set, because each node has their own local model.

- Upon finishing All-reduce all nodes will have the same local model, hence their test losses will match. In general, in IID setting this model will be better performing than individual local models before All-reduce. The reason for the performance improvement is that the aggregate of all local models depends on collective training data, as every local model is trained on its own local dataset, meanwhile in D-PSGD local models are influenced mostly by the data in the neighborhood of the given node, which is a subset of the whole training dataset divided between nodes.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

[3] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[4] Róbert Ormándi, István Hegedűs, and Márk Jelasity. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, 25(4):556–571, 2013.