Project 5: Federated and Distributed Learning

TA: Riccardo Zaccone (riccardo.zaccone@polito.it)

Overview

Modern success of deep learning and the breakthroughs it enables rely on increasingly complex models, whose training requires growing availability of large amounts of data and computing resources. State-of-art architectures, especially those related to NLP, require billions or trillions of parameters, leading to real challenges in training and serving such solutions for real-world applications. In fact, the massive scale of these models leads to prohibitively long training times and makes it impossible to use them on a single general purpose device such as a GPU. Moreover, as hardware speedup scaling laws are already on point, next generation deep learning demands for overcoming the limits of current algorithms and architectures.

To tackle these challenges, distributed methods for deep learning of neural networks have been studied, mainly grouped into *model-parallel* and *data-parallel* methods: the former consists in ways of distributing the computations of a single model splitting its modules across different accelerators, while the latter consists in processing different chunks of data in parallel using multiple accelerators.

Centralized training solutions do not manifest their inadequacies only for learning at scale, but also for leveraging the decentralized nature of data as arising in real-world applications. In practice, the centralized training paradigm no longer matches the needs of modern applications, which need to learn from heterogeneous and massively decentralized data.

Federated Learning (FL) aims to provide a solution for those settings in which privacy is a paramount concern, by constructing a training procedure that offers by design a basic level of privacy. However, statistical and system challenges [3] make the problem of learning from data not only more complex, but also different in terms of assumptions and requirements.

Depending on the setting you are most interested in, you can choose one of the following tracks.

TRACK A: Distributed Learning

1. Goals

In this project we will restrict ourselves to studying data-parallelism approaches. The goal is to become familiar with the objectives of distributed learning of deep neural networks, understand the main challenges and practice with the main approaches in literature. Finally, you will have the opportunity to identify open problems and suggest solutions.

Become familiar with the literature and the state-of-the-art

Before starting the project, you should take some time to study and get familiar with distributed learning, its algorithms, challenges and the main proposed solutions. Here some references:

- **Large Batch Training** [17, 18]
- **Local SGD** [19, 20]
- **Optimizers for distributed learning** [21]

2. Codebase, Resources & Dataset:

To start your experimentation you will first need to implement your codebase. The first step will be to implement a centralized training baseline, to compare the performance of distributed techniques to the standard centralized procedure. Next, extend your code to simulate the parallel training of local methods (*e.g.* algorithm 1 of [19]) by *sequentially* training your workers: this does not change the results of the algorithms, but it is necessary to work with one GPU only.

For the experiments, we require you to use the model architecture described in [11], on the CIFAR-100 dataset, which can be downloaded from torchvision. As per the computational resources, you will use the free version of Google Colab.

Tips and tricks

Good code is initially harder and slower to develop, but it soon repays the initial effort: remind that the code is taken into account as part of the project, and it must be delivered as an attachment to your report. Please adhere to the best coding practices: take into care modularization, avoid deep nesting, mind about readability and lastly use a version control system.

It is in your interest to correctly implement checkpointing and experiment logging. Colab will likely interrupt your experiments, so you must be ready to recover an interrupted run from a certain point in training when resources will be granted again. Not taking this into account will most likely result in (much) more time for carrying out the experiments.

3. Experimentation

Preliminaries

As you may have noticed, the CIFAR-100 dataset you downloaded from torchvision has not a validation split. Your first step will be to split the original dataset such to have a validation set, to be used for hyperparameter tuning. For distributed training, you will also need to have different splits of the training set, so implement a procedure to perform such splits.

Centralized baseline

Train your CNN using SGDM and AdamW optimizers, taking care of searching for the best hyparameters. For both optimizers, perform 150 epochs and use the cosine annealing learning rate scheduler.

Report the plots of training loss, test loss, training accuracy and test accuracy.

Test Large Batch Optimizers

Replace the standard optimizers with their large batch counterparts [17,18]. The suggested way to do it is to subclass the torch.optim.Optimizer class and implement the code for the respective algorithms. It is possible to start from the implementations of SGD and AdamW available in torch. Then, in your codebase, allow the optimizer to be chosen among [SGDM, AdamW, LARS, LAMB] when launching the script (using command line arguments). When experimenting with large-batch optimizers, start from the best hyperparameters found for the base algorithms, and tune the learning rate following [18]. Compare the performance of such optimizers with the one obtained by simply applying the learning rate schedule of [18] to the base optimizers. How do the results change? Up to which batch size is it possible to scale without significantly losing test accuracy?

Switch to Local Methods

Perform an iid sharding of the training set in K={2,4,8} chunks, and train using the LocalSGD algorithm [19]. Explore performing multiple local steps J={4,8,16,32,64}, scaling accordingly the number of iterations (total number of batches processed). For all the experiments, you want to perform as many iterations as in the centralized scenario when increasing the number of local steps or the number of workers (K). Which is the combination that works best? What are the challenges in increasing the number of workers and local steps?

Try other distributed optimizers

There are more advanced solutions that use two optimizers, one for the outer loop and one for the inner one [21]. Extend your experimentation to these solutions. Analyze your results: evaluate the computation and communication cost of each alternative you tried and discuss pros and cons of large batch optimizers and local methods.

4. Time for your personal contribution

Now that you have a clear understanding of the distributed learning scenario and state-of-art solutions, you can now explore a more specific problem. This part of your project is purposely weakly supervised, to let you come up with original ideas. Here we list some of the research directions we believe are most promising.

- **Asynchronous algorithms**: in the above experimentation we limited ourselves to synchronous algorithms. There are some works on asynchronous methods for distributed learning [23, 22, 25] you could start exploring to have an idea of the setting and state-of-art approaches.
- **Optimize for the number of steps in local training methods**: as you may have noticed, there is a tradeoff in increasing the number of local steps. You found the best combination for the task at hand by treating J as a hyperparameter, and tuned on a validation set. A very recent work [26] has delved into the choice of optimal learning rate for LocalSGD, which is based also on the number of local steps. Can you come up with a way to dynamically decide how many steps are to be taken during the training process?

TRACK B: Federated Learning

1. Goals

The goal is to become familiar with the standard federated scenario and its main challenges related to data statistical heterogeneity, systems heterogeneity and their effect on training. You will study the main approaches in literature and finally focus on an open problem and suggest solutions.

Become familiar with the literature and the state-of-the-art

Before starting the project, you should take some time to study and get familiar with the federated scenario, its algorithms and challenges. Here some references:

- **Understand the FL setting** [2, 3, 10]
- **Outlook on issues and state of the art** [7,8,9]
- **Statistical and system heterogeneity:** algorithms [1,4,5] and architectural investigations [12, 13, 14]
- **Real-world applications** [11]

2. Codebase, Resources & Dataset:

To start your experimentation you will first need to implement your codebase. The first step will be to implement a centralized training baseline, to compare the performance of distributed techniques to the standard centralized procedure. Next, extend your code to simulate the parallel training of FL (*e.g.* see [10]) by *sequentially* training your clients: this does not change the results of the algorithms, but it is necessary to work with one GPU only.
For the experiments, we require you to use the model architecture described in [11], on the CIFAR-100 dataset, which can be downloaded from torchvision. For the Shakespeare dataset, you can download it from LEAF [15], and you can use the model architecture described in [2]. As per the computational resources, you will use the free version of Google Colab.

Tips and tricks

Good code is initially harder and slower to develop, but it soon repays the initial effort: remind that the code is taken into account as part of the project, and it must be delivered as an attachment to your report. Please adhere to the best coding practices: take into care modularization, avoid deep nesting, mind about readability and lastly use a version control system.
As specific to deep learning, it is in your interest to correctly implement checkpointing and experiment logging. Colab will likely interrupt your experiments, so you must be ready to recover an interrupted run from a certain point in training when resources will be granted again. Not taking this into account will most likely result in (much) more time for carrying out the experiments.

3. Experimentation

Preliminaries

As you may have noticed, the CIFAR-100 dataset you downloaded from torchvision has not a validation split. Your first step will be to split the original dataset such to have a validation set, to be used for hyperparameter tuning. In FL clients have their own disjoint set of training samples, so to simulate that setting you will need to split the training set into K disjoint subsets. In particular, to simulate statistical heterogeneity, for CIFAR-100 your code should implement the following splittings:

- **iid sharding:** each of K clients is given an approximately equal number of training samples uniformly distributed over the class labels
- **non-iid sharding:** each client is given an approximately equal number of training samples, belonging to N_c classes, where N_c is an hyperparameter you will use to control the severity of the induced dataset heterogeneity. For example, if N_c=1, then each client has samples belonging to one class only.

For the Shakespeare dataset you won't need to perform manual splitting, as the dataset naturally comes in two splits [15].

Centralized baseline

Train your models on both CIFAR-100 and Shakespeare using the SGDM optimizer, taking care of searching for the best hyparameters. As a learning rate scheduler, we suggest you use the cosine annealing scheduler.
Report the plots of test loss and test accuracy. How many epochs do you need? Which learning scheduler performs the best? Explore various solutions and compare your results

The first FL baseline

Implement the algorithm described in [10], fix K=100, C=0.1, adopt an iid sharding of the training set and fix J=4 the number of local steps. Run FedAvg on CIFAR-100 for 2000 rounds respectively, without any learning rate schedule.

The impact of client participation

Choose CIFAR-100, fix K=100, and adopt an iid sharding of the training set. Keep selecting a portion of C=0.1 clients at each round, and simulate two possible client participation schemes:

- **Uniform participation:** each client in the federation has the same probability of being selected at each round. This is the standard setting in FL experiments as done in the previous step.
- **Skewed participation:** each client has a different probability of being selected at each round, and can be used to simulate settings in which some clients are more "active" than others. One way you can implement a skewed sampling is by sampling client selection values according to a Dirichlet distribution parameterized by an hyperparameter γ, and then normalize. In such a way, γ indicates the "severity" of the skewness (the lower γ the higher the heterogeneity of client sampling).

Experiment with different values of ɣ, plot both the resulting sampling distributions and the results of training under such scheduler. Finally, compare the performance of FedAvg to uniform participation (previous step).

Simulate heterogeneous distributions

Fix K=100 and C=0.1, and simulate several non-iid shardings of the training set of CIFAR-100, by fixing the number of different labels clients have (N_c={1,5,10,50}). Then test the performance of FedAvg [10], comparing with the iid sharding, varying the number of local steps J={4,8,16}. When increasing the number of local steps, remember to scale accordingly the number of training rounds.
Is there a noticeable difference in performance? Why does it happen?
Repeat the experiments with the iid and non-iid shards of Shakespeare, with a budget of 200 rounds: how does this compare to the previous results on image classification task? *Hint:* plot the data distributions resulting from your data sharding procedure and the ones of Shakespeare's splits.

4. Time for your personal contribution

Now that you have a clear understanding of challenges in FL, you can now explore a more specific problem. This part of your project is purposely weakly supervised, to let you come up with original ideas. Here we list some of the research directions we believe are most promising.

- **Client selection:** consider a situation in which clients have non-uniform probability of being selected for training at each round. Explore the effect on performance and propose novel methods for addressing such scenarios. You can also take inspiration from the literature (e.g. [24]) to deepen into an aspect of client selection of your choice.
- **Personalized FL:** explore state-of-art solutions for personalized FL [27-32], identify room for improvement and propose novel variations
- **FL-tailored architectures:** some works [12,13] have investigated the role of architectural design in FL. Starting from their findings, propose novel ways of adapting existing architectures to FL training, or novel training paradigms

Deliverables

For the exam, you should deliver:

1. A **report** of your experimentation: it must clearly contain the results of the required experiments, plus whatever is needed to substantiate your personal contribution. Later in the development of the project you can ask more specific feedback about what to include and how to do it.
2. The **codebase** to **reproduce your results**: it should contain a README.md file containing the instructions to run your code to reproduce specific parts of your experimentation. The code itself is not evaluated, but at the exam we may ask for clarifications about some code portions.

🆕 Report guidelines

The report must be written in LateX, using the template available here. It should list all the authors (team members) with associated student ID, in whatever order, and must be long at most 8 pages (references included). Regarding the contents, the report should answer all the questions mentioned in the track description, explaining the motivation with empirical and/or theoretical findings, which must be reported in the report itself. The main part of your report should be devoted to your proposed extension, explaining the methodology (e.g. what you are doing differently from existing approaches to solve the problem you chose) and later the experimental results. Implementation details of the experiments should also be included (e.g. how data has been splitted, how hyperparameter search has been conducted, etc.). You can look at published research papers to have an idea of such a structure.

Common rules

During all the phases of experimentation, remember to always apply the best deep learning practices, such as train-validation-test splitting, performing multiple independent runs for each experiment, ecc.
Before engaging in phase 4, always discuss your plan with the TA.

FAQ

Q: Should I buy online computing resources for doing the project?
A: NO! We don't require you to have access to any computing resource but the free version of Google Colab.

Q: I have a GPU and I prefer using it instead of Google Colab, is it allowed?
A: Yes, if you prefer to use your own resources you sure can. However, we don't require you to have additional computing power than the free version of Google Colab.

Q: Can I avoid proposing and/or implementing some of the points of "Base experimentation"? Can I avoid the "personal contribution" part?
A: The steps detailed in the "Base experimentation" part are all required and represent the bare minimum for the project to be considered of sufficient level. While it is possible to have a sufficient grade developing the first part only, we strongly encourage you to take the challenge of personal contribution, it is the fun part of the project.

Q: I have developed a solution for the "personal contribution" part, but it is not state of the art in terms of results, or it performs worse than expected. Will this be considered as a penalty for the exam?
A: No, we don't require you to advance the state of the art. If your solution performs worse than an existing method, this will not affect the evaluation. We mostly take into account the process of developing your solutions, and marginally take into account how better it is with respect to existing approaches. Being able to discuss your method and results is much more important than having strong results.

Q: I read the suggested papers and I found a lot of math. Do I need to provide theoretical analysis of the method/analysis I propose as a personal contribution?
A: No, we don't require you to perform any theoretical analysis, even if we expect you to know the theoretical framework you're working in (e.g. in FL you should know what the *client drift* is). If you want, your personal contribution can be theoretical, but it must still be supported by properly designed experiments. You can always reach out to the TA for your specific case.

References

[1] Hsu et al., Measuring the effects of non-identical data distribution for federated visual classification, 2019.
[2] Reddi et al., Adaptive federated optimization. ICLR 2021.
[3] Kairouz et al., Advances and open problems in federated learning, Found. Trends Mach. Learn, 2021
[4] Li et al., Federated optimization in heterogeneous networks. PMLR, 2018
[5] Karimireddy et al., Scaffold: Stochastic controlled averaging for federated learning, PLMR 2020
[6] Kairouz et al., Advances and Open Problems in Federated Learning, Foundations and Trends in Machine Learning, 2021
[7] Li at al., A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection, 2019
[8] Chen et al., Privacy and Fairness in Federated Learning: On the Perspective of Tradeoff, ACM computing surveys 2023
[9] Pfeiffer et al., Federated Learning for Computationally Constrained Heterogeneous Devices: A Survey, ACM computing surveys 2023
[10] McMahan et al., Communication-Efficient Learning of Deep Networks from Decentralized Data, AISTATS 2017
[11] Hsu et al., Federated Visual Classification with Real-World Data Distribution, ECCV 2020
[12] Qu et al., Rethinking Architecture Design for Tackling Data Heterogeneity in Federated Learning, CVPR 2022
[13] Pieri et al., Handling Data Heterogeneity via Architectural Design for Federated Visual Recognition, NeurIPS 2023
[14] Li et. al., Fedbn: Federated learning on non-iid features via local batch normalization. ICLR 2021
[15] Caldas et al. "Leaf: A benchmark for federated settings." Workshop on Federated Learning for Data Privacy and Confidentiality 2019
[16] McCandlish, An empirical model of large-batch training, arXiv 2018
[17] You et. Al, Large Batch Training of Convolutional Networks, arXiv 2017
[18] You et. Al, Large Batch Optimization for Deep Learning: Training BERT in 76 minutes, ICLR 2020
[19] Stich et al., Don't Use Large Mini-Batches, Use Local SGD, ICLR 2020
[20] Yu et. Al, On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization, ICML 2019

[21] Wang et. Al, SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum, ICLR 2020
[22] Nabli et al., A2CiD2: Accelerating Asynchronous Communication in Decentralized Deep Learning, NeurIPS 2023
[23] Chen et al., Accelerating Gossip SGD with Periodic Global Averaging, ICML 2021
[24] Nemeth et al., A Snapshot of the Frontiers of Client Selection in Federated Learning, TMLR 2022
[25] SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning
[26] Balles et al., On the Choice of Learning Rate for Local SGD, TMLR 2024
[27] Wang et al., Towards Personalized Federated Learning via Heterogeneous Model Reassembly, NeurIPS 2023
[28] Collins et al., Exploiting Shared Representations for Personalized Federated Learning, ICML 2021
[29] Fallah et a., Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach, NeurIPS 2020
[30] Marfog et al., Personalized Federated Learning through Local Memorization, ICML 2022
[31] Pillutla et al., Federated Learning with Partial Model Personalization, ICML 2022
[32] Shamsian et al., Personalized Federated Learning using Hypernetworks, ICML 2021