Project: Federated Learning Under the Lens of Task Arithmetic

TA: Leonardo Iurada (leonardo.iurada@polito.it)

Overview

Federated Learning (FL) (McMahan et al., 2017) is a paradigm to learn from decentralized data in which a central server orchestrates an iterative two-step training process that involves 1) local training, potentially on a large number of clients, each with its own private data, and 2) the aggregation of these updated local models on the server into a single, shared global model. This process is repeated over several communication rounds. While the inherent privacy-preserving nature of FL makes it well-suited for decentralized applications with restricted data sharing, it also introduces significant challenges.

Model editing refers to the task of modifying, correcting, or improving the functionality of a pre-trained model without retraining it from scratch. A promising direction is **Task Arithmetic** (Ilharco et al., 2023) which involves merging together multiple fine-tuned versions of a base model, each adapted to specific user needs or tasks, to build a single model that integrates this knowledge. This approach naturally aligns with the goals of Federated Learning: both paradigms aim to synthesize distributed expertise into a coherent, performant model while minimizing direct access to raw data. However, a key risk in model merging is *interference*, i.e., the conflict between updates from different sources that degrade overall model quality.

Recent advances have highlighted the emergence of **shared low-sensitivity parameters** (iurada et al., 2025) subsets of model weights that, when fine-tuned, can incorporate new information with minimal interference. This is attributed to the approximate *linearity* of neural functions with respect to weight updates in these sub-spaces, which allows for near-additive composition of fine-tuned models.

The project's main goals are to understand the challenges of FL, specifically when applied on pre-trained vision models and to explore whether leveraging some emergent properties of pre-training can help mitigate such challenges.

Project Specifications

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 pretrained model architecture DINO ViT-S/16 (https://github.com/facebookresearch/dino), 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.
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. Base 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:

- **i.i.d. sharding:** each of K clients is given an approximately equal number of training samples uniformly distributed over the class labels
- **non-i.i.d. 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.

Gentle starting

Centralized baseline

Train your models on CIFAR-100 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

Task Arithmetic

1. Check out the following code snippets to get used to computing parameter sensitivity (i.e., the diagonal elements of the Fisher Information matrix) scores and calibrate gradient masks in multiple rounds:
https://github.com/iurada/talos-task-arithmetic/blob/ae102473de0e57ebf625eca22e10781c371149ec/vision/pruners.py#L203
https://github.com/iurada/talos-task-arithmetic/blob/main/vision/prune_finetune.py#L107
2. By extending SGDM, implement your SparseSGDM optimizer that accepts as input also a gradient mask to zero-out the updates of the weights whose corresponding entry in the mask is zero. Here you can see the SGDM pseudo-code, for reference:
https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

More info can be found in [15] (**HINT:** see pseudo-code in the Appendix and github repo).

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 a proper number of rounds (up to you to define, based on convergence and time/compute budget).

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? Motivate your findings.

Task Arithmetic techniques in FL

Sparse fine-tuning consists in updating only a selected subset of parameters by eg. masking the gradients during GD. This is typically done by:
(step 0.: Either train or obtain in closed-form eg. Ridge, a classifier locally which will be then kept frozen at all times after this step)

1. **Calibrate a gradient mask**, i.e., decide which weights will be updated (entry in the mask == 1) and which not (entry in the mask == 0).
   Calibrate the gradient mask by identifying **in multiple rounds** (see why in [15], Sec. 4.2.) the least-sensitive parameters (i.e. the weights with a sensitivity score lower than some user-defined threshold).
2. **Perform fine-tuning by masking gradients** with the calibrated masks. Use your implementation of SparseSGDM.

You should now experiment with:
- Sparsity ratio (defined by the sensitivity threshold chosen in step 1.)
- Number of calibration rounds

4. Extension

Now that you have a clear understanding of challenges in FL, you can now explore a more specific problem. You can either:

- **[Guided Extension]**
  Try different gradient mask calibration rules (other than least-sensitive parameters). Specifically, you should try and compare with the previous results:
  1. Pick the most-sensitive weights (instead of the least-sensitive)
  2. Pick the lowest-magnitude weights (instead of the least-sensitive)
  3. Pick the highest-magnitude weights (instead of the least-sensitive)
  4. Pick random weights (instead of the least-sensitive)

Or propose your own novel contribution. This part of your project is purposely left as a **[Weakly Supervised Extension]**, to let you come up with original ideas. ***<u>In this case, before engaging with your own idea, please discuss your plan with the TA.</u>*** Some directions could be:

- An additional analysis of some aspects of federated learning (FL) or task arithmetic not included in the above specifications
- Original use of task arithmetic techniques in federated learning, with the purpose of addressing the challenges outlined above
- New task arithmetic techniques inspired by the connections with FL and compatible with FL constraints

IMPORTANT:
The score for your chosen Extension will depend on the quality of its execution. Even a strong idea will not receive high marks if it is poorly implemented. Therefore, before making your choice, consider how much time you have available. If time is limited, it is advisable to select the Guided Extension, as completing it thoroughly can still earn you full marks.

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 (excluding references). Regarding the contents, the report should follow standard research communication practices (e.g., divide the report into: title, abstract, introduction, related works, methodology, experiments, conclusion…) and 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.

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.

Q: The main idea is to create a centralized model that will be the benchmark for the Federated Learning approach, and we also need to consider both cases of iid and non-iid sharding, and on top of that also the task arithmetic process. So in total we need to develop 6 models:
- Centralized (benchmark);
- Federated iid;
- Federated non-iid;
- Centralized with task arithmetic (benchmark);
- Federated iid with task arithmetic;
- Federated non-iid with task arithmetic.

A: TODO

Q: Can we use the Flower framework?
A: Yes, you can use any torch-based framework for your project. More in general, you can also use any public code, as long as you are completely aware of what any piece of code is doing in your project. It is possible to get answers on your code during the oral exam, so it is important to be able to explain any piece of code you have used, even if that code comes from another codebase.

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] Iurada et al., Efficient Model Editing with Task-Localized Sparse Fine-tuning. ICLR 2025.