Projet de fin d'étude à l'ISAE-Supaero.
Filière Sciences des Données et de la Décision.
Domaine Energie, Transport et Environnement.
Master Mathématiques et Applications, parcours Recherche Opérationnelle.

Master's Thesis
Internship Report

Data Summarization Methods for Energy Efficient Federated Learning

Author :
Julianna DEVILLERS

Supervisors :
Mr. Olivier BRUN
Mr. Balakrishna PRABHU

October 6, 2023

Contents

List of Figures

List of Tables

List of Acronyms

CNRS *French National Centre for Scientific Research.*

LAAS *Laboratory of Analysis and Architecture of Systems.*

SARA *Services and Architectures for Advanced Networks.*

CNN *Convolutional Neural Network.*

DL *Deep Learning.*

DNN *Deep Neural Network.*

DS *Data Summary.*

FL *Federated Learning.*

IID *Independent and Identically Distributed.*

LSTM *Long Short-Term Memory.*

ML *Machine Learning.*

MLP *Multilayer Perceptron.*

NLP *Natural Language Processing.*

NN *Neural Network.*

RPC *Remote Procedure Call.*

SGD *Stochastic Gradient Descent.*

Chapter 1

Introduction

1.1 Federated Learning

Machine learning is becoming increasingly important due to the growing volumes and variety of data, but also due to the easy access to increasing computing power. In particular, classical machine learning involves centralized data learning, where the data are collected and the entire learning process is executed on a central server. Today, the models developed are capable of analyzing large and complex datasets very quickly and accurately. But as datasets become too large to be stored on or processed by a centralized server, training the models increasingly requires dividing the tasks over multiple machines in order to parallelize computing power. Distributed deep learning splits the workload into many subproblems that require less storage and solving time and that can be assigned to different devices. In traditional distributed learning, a common assumption is that the local datasets are independent and identically distributed (i.i.d.). This can happen for example when the data are distributed from the central server to the local nodes.

However, this is not always the case. As edge devices such as cell phones, wearable devices or vehicles generate a multitude of data every day, it can be difficult, for reasons of network bandwidth limitations, device availability or storage capability, to gather all the data from the edge devices in the data center and redistribute it. In addition, for data privacy reasons, it may not be possible to share the local data of a device. In this context, being able to collaboratively train a shared global model from decentralized private datasets becomes interesting. This is the idea behind Federated Learning (FL), a paradigm undergoing an explosive growth of interest in a wide range of fields. FL has a wide range of applications, especially for sensitive data for example in the fields of healthcare or finance or considering data on people localization. At present, it has been widely used to optimize the experience of smartphone users while protecting privacy. For example, Google applied FL for device setting recommendation in the Pixel mobile phone [ai.18]. It also used it for keyboard suggestions in Gboard [HRM+19] and Android Message [sup23], and so did Apple [PSM+21][Hao19].

Federated Learning comes of course with its own challenges, a major one being the heterogeneity in the networks. Furthermore, a high number of communication rounds between the central server and the remote devices can be needed to achieve good performance which can be expensive.

1.2 The question of Energy Cost

Training state-of-the-art deep learning models requires a large number of experiments to be run in practice. This translates into large amounts of computing and energy usage [SGM19]. Take the example of the GPT-3 language model (175 billion parameters), developed by OpenAI. Training this model consumed around 1,287 MWh of energy, corresponding to 552 TeqCO2 [PGL+21]. In comparison, 1TeqCO2 is equivalent to a round trip from Paris to New York by plane, and in France the figure of 2TeqCO2 is also regularly mentioned as the annual carbon budget per person to be achieved by 2050. GPT-3 training therefore corresponds to 276 times this per capita carbon budget. Federated learning does not need energy-consuming cooling systems, unlike centralized data centers, and can benefit from the reduced energy consumption of embedded devices. However, depending on the configuration, FL training can also require a lot of computation and communication. [QPFM+23] found that in certain settings, CO2 emissions can be up to two orders of magnitude higher for FL training compared with centralized. As FL is becoming more and more prevalent, how to reduce its energy footprint becomes a real issue.

1.3 Working Environment

1.3.1 LAAS-CNRS

The Laboratory of Analysis and Architecture of Systems (LAAS) is a CNRS research unit located in Toulouse and currently directed by Mohamed Kaâniche. It is linked with the Institute for Engineering and Systems Sciences (INSIS) and the Institute of Information Sciences and their interactions (INS2I).

Created in 1968 by Jean Lagasse in association with the University of Toulouse under the name of "Laboratoire d'automatique et de ses applications spatiales", the LAAS-CNRS had over 577 employees on January 1, 2019, including researchers and teacher-researchers, PhD students and ITAs (engineers and technical research staff). To date, the teams have supervised over 2,000 theses and won 14 CNRS awards. The premises include 8 technology platforms for carrying out various projects requiring powerful experimental facilities: a clean room, a robots platform, a characterization platform, a design platform, an energy platform, a micro and nanotechnologies platform, an IT engineering platform and an embedded systems platform for space.

Research activities at LAAS-CNRS focus on 4 disciplines which have been the hallmark of the laboratory since its inception (computer science, robotics, automatic control and micro and nanosystems) and 5 transversal application axes (energy, space, smart factories, health/environment, transport/mobilities). Within these disciplines, research at LAAS revolves around 6 scientific departments running 26 research teams, basic units of research at the laboratory. The organisation chart is provided in Figure 1.1. The internship was carried out within the RISC Department on "Trustworthy Computing Systems and Networks" and the Services and Architectures for Advanced Networks (SARA) team. The research activities in the SARA team focus on next generation networks and communication systems, as well as their applications.

1.3.2 Internship Topic

The internship was part of the ANR DELIGHT project, whose aim is to provide a framework for evaluating the energy consumption of FL algorithms as well as to propose new algorithms with improved energy-efficiency. The internship topic fits in

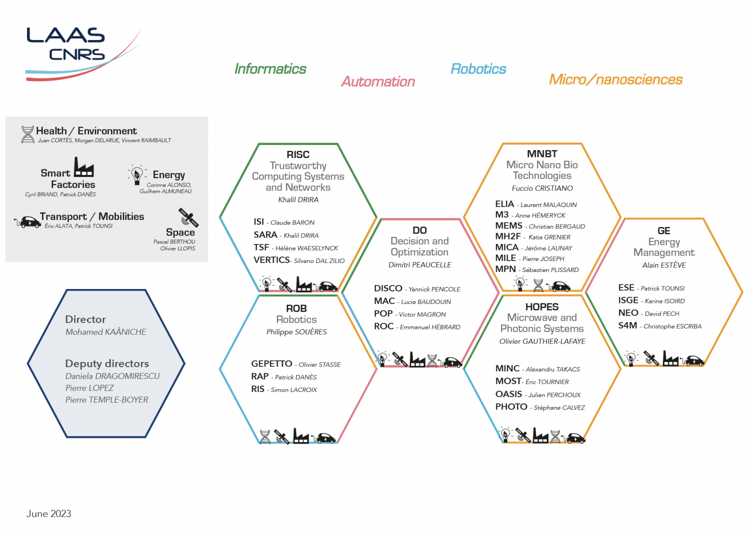


Figure 1.1: Organisation chart of LAAS-CNRS teams. June 2023.

the second objective of the DELIGHT project. It was supervised by Olivier Brun and Balakrishna Prabhu.

Initially, the topic of the internship aimed at focusing on techniques for reducing FL energy consumption including data summarization and gradient compression. On reconsideration, only data summarization was studied. A PhD thesis will follow the works conducted here, using other methods such as speed-scaling and models for energy consumption for the nodes. The aim of the thesis will be to exploit the heterogeneity of data in FL to reduce the energy footprint.

The internship began with a study of the existing research works in federated learning and data summarization. This literature review, presented in section 2, goes over the various issues and algorithms used for FL, as well as the use of submodular functions to produce a data summary. After an initial theoretical study of the convergence linked to the use of a data summary in the context of FL, we focused on the implementation of a federated system and data summaries in order to experimentally test the relevance of the proposed technique.

Chapter 2

Related Work

2.1 Federated Learning Algorithms

Federated learning (FL) [MMR$^+$16] enables multiple actors to build a common, robust machine learning model without sharing data, thus allowing to address critical issues such as data privacy, data security, data access rights and access to heterogeneous data. Federated Learning is a distributed learning paradigm with two key challenges that differentiate it from traditional distributed optimization: (1) significant variability in terms of the systems characteristics on each device in the network (systems heterogeneity), and (2) non-identically distributed data across the network (statistical heterogeneity).

Statistical heterogeneity is the main challenge in FL where clients have different data distribution that can lead to model bias during training. Many existing works address this issue. The most representative is FedAvg [MMR$^+$16], which, as the first FL optimization algorithm, proposes to cooperatively train a global model by transferring model parameters of clients. The edge server averages the received model parameters as global model parameters and then sends them back to local clients for the next global round. However, local clients are usually updated towards the local optimum during local training while the global model is updated towards the global optimum, which can lead to a gap in the optimal point between local models and the global model.

Therefore, FedProx [LSZ$^+$18] proposes a regularization term to correct this gap. When updating locally, different clients can execute different local steps proposed by FedNova [WLL$^+$20]. An asynchronous layer is introduced by FedAT [CCA$^+$21] in which the clients are clustered based on the performance of local training. Personalized federated learning is another extensively studied topic where each client has its own specialized model rather than a shared global model. Hereinafter, one strategy of base layers plus personalized layers is proposed by [AASC19], which only updates the base layers during federated training, and then clients update their personalized layers based on their own local data with base layers fixed. PFedMe is proposed by [DTN20] where clients can update their local models in different directions while not deviating from a global reference point. Moreover, according to different learning tasks among clients, the strategy of dividing clients into different clusters and then aggregation within the clusters is proposed in [GCYR20, GHYR19], thus avoiding deviations when global model aggregates. We propose in this section a succinct non-exhaustive but insightful review of major FL approaches.

2.1.1 Federated Averaging

The general learning problem considered in [MMR+16] is

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w), \tag{2.1}$$

where n is the number of training samples and $f_i(w) = \ell(x_i; y_i, w)$ is the loss of the prediction on example $(x_i; y_i)$ made with the model parameters w. In the *Federated Learning* setting, it is assumed that there are K clients over which the data are partitioned, with $\mathcal{P}_k$ the set of indexes of data points on client k, with $n_k = |\mathcal{P}_k|$. Thus, the objective (2.1) can be rewritten as

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \text{ where } F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \tag{2.2}$$

If the partition $\mathcal{P}_k$ was formed by distributing the training examples over the clients uniformly at random, then we would have $\mathbb{E}_{\mathcal{P}_k}[F_k(w)] = f(w)$, where the expectation is over the set of examples assigned to a fixed client k. This is the IID assumption typically made by distributed optimization algorithms; Federated learning has however several key properties that differentiate it from a typical distributed optimization problem:

- **Non-IID**: The training data $\mathcal{P}_k$ on a given client k are typically based on the usage of the mobile device by a particular user, and hence any particular user's local dataset will not be representative of the population distribution.

- **Unbalanced**: Similarly, some users will make much heavier use of the service or app than others, leading to varying amounts n_k of local training data.

- **Massively distributed**: the number of clients K participating in an optimization is much larger than the average number $\sum_k n_k/K$ of examples per client.

- **Limited communication**: Mobile devices are frequently offline or on slow or expensive connections. In federated learning schemes, additional computations are performed in order to reduce the number of rounds of communication needed to train a model. This is achieved either by increasing the parallelism (more clients working independently between each communication round) or by increasing the computation on each client (e.g., several gradient updates between each communication round).

2.1.1.1 FederatedSGD

The usual approach for minimizing the objective function (2.2) is based on synchronous Stochastic Gradient Descent (SGD). FederatedSGD (or FedSGD), the baseline algorithm considered in [MMR+16], selects a fraction C of clients on each round, and computes the gradient of the loss over all the data held by these clients. Thus, C controls the global batch size, with $C = 1$ corresponding to full-batch (non-stochastic) gradient descent. A typical implementation of FedSGD with $C = 1$ and a fixed learning rate η has each client k compute $g_k = \nabla F_k(w_t)$, the average gradient on its local data at the current model w_t, and the central server aggregates these gradients and applies the update

$$w_{t+1} \leftarrow w_t - \eta \nabla f(w_t) = w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k.$$

An equivalent scheme is given by updating the model parameters locally on each client

$$w_{t+1}^k \leftarrow w_t - \eta g_k, \quad \text{for all } k = 1, \ldots, K,$$

and then averaging the model parameters on the central server

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k.$$

That is, each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models.

2.1.1.2 FedAvg

We can add more computation to each client by iterating the local update $w^k \leftarrow w^k - \eta \nabla F_k(w^k)$ multiple times before the averaging step. This approach is termed FederatedAveraging (or FedAvg). The amount of computation is controlled by three key parameters: C, the fraction of clients that perform computation on each round; E, the number of training passes each client makes over its local dataset on each round; and B, the local minibatch size used for the client updates. Note that FedSGD corresponds to $B = \infty$ (that is, full local dataset is treated as a single minibatch) and $E = 1$. Complete pseudo-code of FedAvg is given in Algorithm 1. The convergence of FedAvg has been analysed in several works [LHY+20][YYZ18][KKM+20] and the algorithm has established itself as the algorithm of choice for federated learning due to its simplicity and relatively low communication cost. Most FL methods are actually variants of the FedAvg algorithm.

2.1.2 FedProx

While FedAvg has demonstrated empirical success in heterogeneous settings, it does not fully address the underlying challenges associated with heterogeneity. In the context of systems heterogeneity, FedAvg does not allow participating devices to perform variable amounts of local work based on their underlying systems constraints; instead it is common to simply drop devices that fail to compute E epochs within a specified time window. From a statistical perspective, FedAvg has also been shown to diverge empirically in settings where the data are non-identically distributed across devices.

The authors of [LSZ+18] observe that the number of local epochs E in FedAvg plays an important role in convergence. On one hand, performing more local epochs allows for more local computation and potentially reduced communication, which can greatly improve the overall convergence speed in communication-constrained networks. On the other hand, with dissimilar (heterogeneous) local objectives F_k, a larger number of local epochs E may lead each device towards the optima of its local

Algorithm 1 FedAvg algorithm

```
 1: procedure SERVER($C, E, B$)                         ▷ Run on central server
 2:     Initialize $w_0$
 3:     for each round $t = 1, 2, \dots, T$ do
 4:         $m \leftarrow \max(C\,K, 1)$
 5:         $S_t \leftarrow$ random set of $m$ clients
 6:         $N_t \leftarrow \sum_{k \in S_t} n_k$
 7:         for each client $k \in S_t$ in parallel do
 8:             $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
 9:         end for
10:         $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{N_t} w_{t+1}^k$
11:     end for
12:     return $w_{T+1}$
13: end procedure
14: procedure CLIENTUPDATE($k, w$)                      ▷ Run on client $k$
15:     $\mathcal{B} \leftarrow$ split $\mathcal{P}_k$ into batches of size $B$
16:     for each local epoch $i = 1, 2, \dots, E$ do
17:         for batch $b \in \mathcal{B}$ do
18:             $w \leftarrow w - \eta \nabla \ell(w; b)$
19:         end for
20:     end for
21:     return $w$ to server
22: end procedure
```

objective as opposed to the global objective – potentially hurting convergence or even causing the method to diverge. Further, in federated networks with heterogeneous systems resources, setting the number of local epochs to be high may increase the risk that devices do not complete training within a given communication round and must therefore drop out of the procedure.

There is therefore an interplay between systems and statistical heterogeneity in federated learning. To address this issue, FedProx proposes adding a proximal term to the objective that helps to improve the stability of the method. The objective function of client k is modified as follows

$$h_k(w, w_t) = F_k(w) + \frac{\mu}{2}\|w - w_t\|^2. \tag{2.3}$$

The proximal term is beneficial in two aspects: (1) It addresses the issue of statistical heterogeneity by restricting the local updates to be closer to the initial (global) model w_t without any need to manually set the number of local epochs. (2) It allows for safely incorporating variable amounts of local work resulting from systems heterogeneity by minimizing $h_k(w, w_t)$ up to a precision γ_k^t which can be adapted to the processing capacity of each client k. Note also that from a theoretical point of view, the coefficient μ can be chosen sufficiently large for the local problems to be convex, which eases the convergence analysis (see [LSZ$^+$18]).

2.1.3 FedNova

The clients participating in federated learning are typically highly heterogeneous, both in the size of their local datasets as well as their computation speeds. In FedAvg each client performs E epochs of local-update stochastic gradient descent (SGD) with a mini-batch size B. Thus, if a client has n_k local data samples, the number of local

Algorithm 2 FedProx algorithm

```
1: procedure SERVER($m$)                                    ▷ Run on central server
2:     Initialize $w_0$
3:     for each round $t = 1, 2, \dots, T$ do
4:         Select a subset $S_t$ of $m$ clients at random
5:         $N_t \leftarrow \sum_{k \in S_t} n_k$
6:         for each client $k \in S_t$ in parallel do
7:             $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
8:         end for
9:         $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{N_t} w_{t+1}^k$
10:    end for
11:    return $w_{T+1}$
12: end procedure
13: procedure CLIENTUPDATE($k, w_t$)                         ▷ Run on client $k$
14:    Find $w_k^{t+1}$ which is a $\gamma_k^t$-inexact mimimizer of $h_k(w, w_t) = F_k(w) + \frac{\mu}{2}\|w - w_t\|^2$
15:    return $w_k^{t+1}$ to server
16: end procedure
```

SGD iterations is $\tau_k = \lfloor E n_k / B \rfloor$, which can vary widely across clients. Moreover, as the clients' computing speeds are heterogeneous, faster clients can perform more local updates than slower clients. The authors of [WLL+20] observe that the heterogeneity in local updates causes objective inconsistency. More precisely, standard averaging of client models after heterogeneous local updates results in convergence to a stationary point – not of the original objective function $f(w)$, but of an inconsistent objective $\tilde{f}(w)$, which can be arbitrarily different from $f(w)$ depending upon the relative values of τ_i. The idea is illustrated in Figure 2.1.

The authors of [WLL+20] also propose FedNova, a method that correctly normalizes local model updates when averaging. The main idea of FedNova is that instead of averaging the cumulative local gradient returned by client k (which performs τ_k local updates) in t-th training round with $w_{t+1} = \sum_{k \in S_t} \frac{n_k}{N_t} w_{t+1}^k$, the aggregator averages the normalized local gradients with $w_{t+1} = w_t + \tau_{eff} \sum_{k \in S_t} \frac{w_{t+1}^k - w_t}{\tau_k}$ where τ_{eff} has to be chosen to scale up or scale down the effect of the aggregated updates (as for a global learning rate). The details of the convergence analysis are in [WLL+20].

2.1.4 FedOpt

The FedAvg update is:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k = w_t - \sum_{k=1}^{K} \frac{n_k}{n}(w_t - w_{t+1}^k).$$

[RCZ+21] observe that this server update is equivalent to applying SGD with a learning rate of 1 to the "pseudo-gradient" $-\Delta_{t+1}$ where $\Delta_{t+1}^k = w_{t+1}^k - w_t$ and $\Delta_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \Delta_{t+1}^k$. Based on that, they introduce other formulations where SGD is replaced by adaptive optimizers (e.g. Adam, AdaGrad, etc). They show that their method can significantly improve the performance of FL, especially for non-iid data.

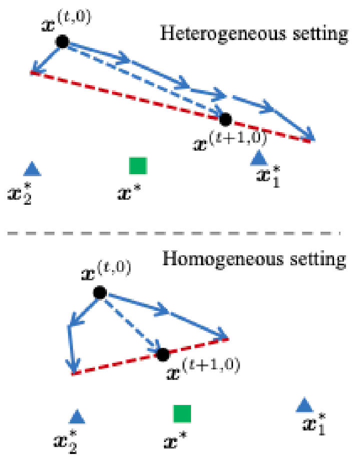


Figure 2.1: FedAvg model updates in the parameter space for a single epoch of training. Green squares and blue triangles denote the minima of global and local objectives, respectively. [WLL+20]

2.1.5 PFNM and FedMA

[YAG+19] observe that the ordering of neurons of a hidden layer of a multilayer perceptron (MLP) is permutation invariant meaning that weights could be treated as unordered collections of vectors instead of matrices. More generally, for any given neural network (NN), many variants of it that differ only in the ordering of those neurons would achieve the same performance. For example, consider two fully-connected layers of a NN A_i and A_{i+1} and P a permutation matrix. Given O the output to the $i+1$ layer and I the input of the i-th layer. We have $O = A_{i+1} \cdot A_i \cdot I = A_{i+1} \cdot P \cdot P^{-1} \cdot A_i \cdot I$ so the output would be the same with $A_{i+1} \cdot P$ as the weights for the layer $i+1$ and $P^{-1} \cdot A_i$ for the layer i. More details on permutation invariance for different architectures are provided in [WYS+20].

Due to this permutation invariance, the j-th neuron of a layer in an MLP is unlikely to correspond to the neuron with the same index of a different MLP therefore preventing a coordinate-wise averaging of weights such as what was proposed with FedAvg to achieve good performance. To address this problem, [YAG+19] introduces Probabilistic Federated Neural Matching (PFNM), an approach that matches the neurons of client neural networks before averaging them, using Bayesian non-parametric methods. As a result, PFNM has better performance and communication efficiency than FedAvg on fully connected feedforward networks. The permutation invariance problem also arises with CNN and LSTM architectures and [WYS+20] extends the method to those architectures by introducing FedMA.

Unlike classical FL methods which require that all clients share the same features space and model architectures, PFNM and FedMA allow for more heterogeneity while helping prevent weight mismatching problems.

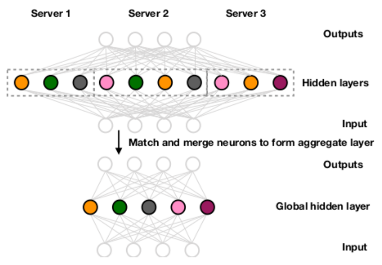


Figure 2.2: Single layer Probabilistic Federated Neural Matching algorithm showing matching of three MLPs. Nodes in the graphs indicate neurons, neurons of the same color have been matched. [YAG+19]

2.1.6 FedAT

One of the main challenges of Federated Learning is the straggler problem where clients lag due to data or resource heterogeneity or even go offline. Not integrating the slower clients can prevent the model from converging to the optimal solution. FedProx and FedNova both propose solutions where the server waits for the slower clients to update, which can lead to significantly prolonged training time. Other existing solutions use asynchronous approaches [XKG20] [CNSR20] [LHD+20] where the server can aggregate without waiting for the straggling clients. However, as the server communicates with all the clients asynchronously, a huge number of clients can update the model simultaneously therefore creating a communication bottleneck. To overcome this, [CCA+21] proposes FedAT, combining synchronous and asynchronous FL training using a tiering mechanism. Furthermore, asynchronous approaches suffer from high communication costs as they require much more frequent communications between clients and the server. To minimize the communication cost introduced by asynchronous training, FedAT uses Encoded Polyline Algorithm to compress the model data transferred between the clients and the server.

2.1.6.1 Training Process

FedAT consists of three main components: the centralized server for global model synchronization, the clients, and a tiering module that partitions clients into different performance tiers based on their response latency (i.e., the time they take to finish a single round of training). For M tiers where $tier_1$ is the fastest, the server maintains a list of M models $w^t_{tier_i}$, one for each tier which correspond to the most updated view of local models, at a certain round t. It also maintains a global model w asynchronously updated from the tiers. The process combines intra-tier synchronous training with cross-tier asynchronous training and is illustrated in Figure 2.3.

2.1.6.2 Weighted Aggregation

For intra-tier training, FedAT uses the surrogate objective function $h_k(w_k, w)$ for client k that was proposed in FedProx where w_k, w are the local model of client k and server model, respectively.

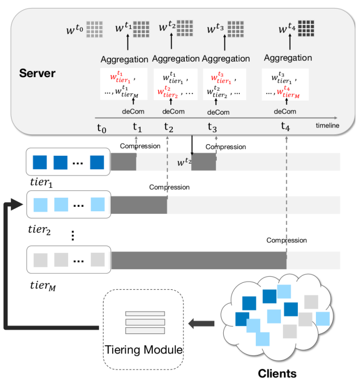


Figure 2.3: Overview of FedAT training process. deCom denotes the decompression process of clients' models in a certain tier on the server. [CCA+21]

Given S_t a subset of randomly selected clients in tier m, N_c the total number of data samples in S_t and n_k the number of local training data of a client k, the update of $f_{tier_m}(w)$, the weighted average of the models from the selected clients, is as follows:

$$f_{tier_m}(w) = \sum_{k=1}^{|S_t|} \frac{n_k}{Nc} h_k(w_k) \tag{2.4}$$

Uniformly aggregating the asynchronously updated tier model into the global model may cause the global model to bias towards the faster tiers as previously illustrated in Figure 2.1. Instead, FedAT proposes a weighted aggregation heuristic, that assigns higher weight to slower tiers. That weight is computed based on the number of times a tier has updated the global mode. Given the number of updates from each tier till now is $T_{tier_1}, T_{tier_2}, \ldots, T_{tier_M}$ respectively and $T = T_{tier_1} + T_{tier_2} + \ldots + T_{tier_M}$. The global objective function of FedAT is defined as:

$$f(w) = \sum_{m=1}^{M} \frac{T_{tier_{M+1-m}} T}{f}_{tier_m} (w) \tag{2.5}$$

This way, slower tiers are assigned larger weight values as, for a slower tier, $M + 1 - m$ corresponds to a relatively faster tier with a higher $T_{tier_{M+1-m}}$. The pseudo-code for FedAT is given in Algorithm 3.

2.1.7 FedPer

As previously stated, statistical heterogeneity is the main challenge in FL. [AASC19] focuses on specific personalization tasks. As same input data may receive different labels from different users, these tasks are first characterized by the need of different models across users. Therefore, traditional FL procedures achieve poor performance on those tasks. However, users have insufficient data individually to efficiently train an isolated model thereby making FL approaches relevant. The authors of [AASC19] introduce FedPer which consists of base layers trained using FL algorithms and personalization layers locally trained with traditional machine learning methods (such as stochastic gradient descent). This method, illustrated in Figure 2.4, can better capture statistical heterogeneity and help alleviate its ill-effects.

Algorithm 3 FedAT algorithm

```
1:  procedure SERVER($m$)                                              ▷ Run on central server
2:      Initialize $w_{tier_1}, w_{tier_2}, \ldots, w_{tier_M}$ to $w^{t_0}$. Initialize $t, T_{tier_1}, \ldots, T_{tier_M}$ to 0
3:      for each tier $m \in M$ in parallel do
4:          while $t < T$ do
5:              $w_t \leftarrow \text{WeightedAverage}(w_{tier_1}, w_{tier_2}, \ldots, w_{tier_M})$
6:              Select a subset $S_m$ of clients from tier $m$ at random
7:              for each client $k \in S_m$ in parallel do
8:                  $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
9:              end for
10:             $N_c \leftarrow \sum_{k \in S_m} n_k$
11:             $w_{tier_m} \leftarrow \sum_{k \in S_m} \frac{n_k}{N_c} \cdot w_{t+1}^k$
12:             $T_{tier_m} \leftarrow T_{tier_m} + 1$
13:             $t \leftarrow t + 1$
14:         end while
15:     end for
16:     return $w_{T+1}$
17: end procedure

18: procedure CLIENTUPDATE($k, w_t$)                                     ▷ Run on client $k$
19:     $w_k^{t+1} \leftarrow w_k^t - \eta \nabla h(w^t)$
20:     return $w_k^{t+1}$ to server
21: end procedure

22: procedure WEIGHTEDAVERAGE($w_{tier_1}, w_{tier_2}, \ldots, w_{tier_M}$)    ▷ Run on central
    server
23:     if t==0 then
24:         return $w^{t_0}$
25:     else
26:         $T \leftarrow \sum_{m \in M} T_{tier_m}$
27:         return $\sum_{m=1}^{M} \frac{T_{tier_{M+1-m}}}{T} \cdot w_{tier_m}$
28:     end if
29: end procedure
```

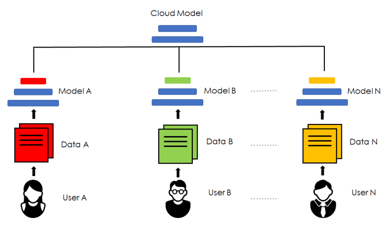


Figure 2.4: Overview of FedPer approach. All users devices have the same base layers (colored blue) that are updated from the server at each iteration. [AASC19]

2.1.8 PFedMe

[DTN20] proposes pFedMe, another method to handle statistical diversity and achieve personalization in FL, where clients can pursue their own models with different directions while staying close to the global model w. PFedMe aims to solve a bi-level problem. The inner problem of pFedMe consists in obtaining an optimal personalized model θ_i for each client i. With f_i the expected loss over the data distribution of the client i and λ a regularization parameter, the inner problem is :

$$F_i(w) = \min_{\theta_i} \Big\{ h_i(\theta_i, w) := f_i(\theta_i) + \frac{\lambda}{2} ||\theta_i - w||^2 \Big\} \tag{2.6}$$

At the outer level, pFedMe aims to solve:

$$\min_{w} \Big\{ F(w) := \frac{1}{N} \sum_{i=1}^{N} F_i(w) \Big\} \tag{2.7}$$

The algorithm proposed for pFedMe deals with the outer problem by using gradient descent with respect to F_i instead of f_i and introduces an additional parameter β for the global model update. Algorithm 4 provides the pseudo-code of pFedMe, the tilde over variables designating approximations detailed in [DTN20].

Algorithm 4 PFedMe algorithm

```
 1: procedure SERVER($m$)                                  ▷ Run on central server
 2:     Initialize $w_0$
 3:     for each round $t = 1, 2, \dots, T$ do
 4:         Select a subset $S_t$ of $m$ clients at random
 5:         for each client $k \in S_t$ in parallel do
 6:             $w^k_{t+1} \leftarrow \text{ClientUpdate}(k, w_t)$
 7:         end for
 8:         $w_{t+1} \leftarrow (1-\beta)w_t + \beta \sum_{k \in S_t} \frac{w^k_{t+1}}{m}$
 9:     end for
10:     return $w_{T+1}$
11: end procedure
12: procedure CLIENTUPDATE($k, w_t$)                        ▷ Run on client $k$
13:     $w^k_{t,0} = w_t$
14:     for r = 0 to R-1 do
15:         Sample a fresh mini-batch $B$ and find $\tilde{\theta}_i$ which is a $\delta$-approximate minimizer of $\tilde{h}_i(\theta_i, w^k_{t,r}, B)$
16:         $w^k_{t,r+1} \leftarrow w^k_{t,r} - \eta \nabla \tilde{F}_i(w^k_{t,r})$
17:     end for
18:     return $w^k_{t,R}$ to server
19: end procedure
```

2.1.9 Other multi-model approaches

Multi-model techniques can be particularly relevant for non-IID data, since they enable personalized or device-specific modeling therefore being a lever to handle the statistical heterogeneity of the data.

In particular, multi-task learning techniques aim to learn models for multiple related tasks simultaneously. For federated learning, different tasks could for example correspond to different data distributions. [SCST18] first extended multi-task learning to FL introducing MOCHA in which at each iteration, either the weights for each task or the relationships amongst the tasks are being optimized. While MOCHA has been demonstrated effective, it works only for convex objectives. [CB19] proposes VIRTUAL to handle non-convex objectives, constructing a star-shaped Bayesian network representing the central server and the clients and performing variational inference during learning. However, the main limitation to most multi-task methods for FL is that they are expensive to generalize to massive networks and generally assume that all clients participate in each training round. Other multi-task approaches exist. For example, [EKM+19] observes that, when considering that the devices are users mobile phones, the devices are more likely to be available and meet training eligibility requirements during night. Therefore, the authors of [EKM+19] proposed a pluralistic multi-task approach to address this cyclic structure.

Clustering-based methods are another way to deal with the data heterogeneity issue in FL by partitioning clients into different clusters based on their data distribution or other characteristics. [SMS21] proposed Clustered Federated Learning, a method that group clients into clusters based on the cosine similarity between gradient updates in order to identify the clients that can be trained together. [GCYR20] introduced the Iterative Federated Clustering Algorithm (IFCA), where at each iteration, clients identify the cluster they belong to before running local updates and sending them to update the model parameters for each cluster.

Meta-learning can be considered as learning to learn and can also be applied to federated learning. Works [JKRK19][KBT19][LKCT19] have shown that it is also a relevant framework to model the personalization objectives for FL in the context of non-IID data.

2.1.10 Convergence analysis

The main challenge in FL being the statistical heterogeneity, the analysis techniques used for centralized learning must be adapted for the non-IID case by adding assumptions on data dissimilarities. A number of works have studied the convergence of FL algorithms, trying to relax the less realistic assumptions. For example, the authors of [LHY+20] propose a convergence analysis for FedAvg where only a subset of clients participate in the training at each round. The authors of [YL22] note that the assumption taken to bound gradient dissimilarity in the work introducing FedProx [LSZ+18] is quite unrealistic and propose to relax this assumption and extend the analysis to loss functions that are not smooth. However, they assume that the local client functions are L-Lipschitz which is also a restrictive condition on gradient dissimilarity. In Table 2.3 are some examples of convergence results along with the assumptions taken for FedAvg and FedProx. The explanations of the assumptions are in Table 2.1 and Table 2.2. These tables are adapted from the work of [KMA+21]. We note here T the total number of communication rounds, E the number of local rounds between each communication round, K the total number of clients, S the number of clients participating in each round.

Table 2.1: Non-IID assumptions.

Symbol	Full name	Explanation
(G, B)-BGD	Bounded gradient dissimilarity	$\mathbb{E}_k[\|\|\nabla F_k(w)\|\|^2] \leq G^2 + \|\|\nabla f(w)\|\|^2 \cdot B^2$
BCGV	Bounded inter-client gradient variance	$\mathbb{E}_k[\|\|\nabla F_k(w) - \nabla f(w)\|\|^2] \leq \delta^2$

There are of course other significant works and approaches covering the topic of FL due to the growing interest in FL, both in industry and research. The goal of this section was however to explain some of the main concepts in FL and show the different challenges arising in the FL setup. Some of the methods covered in this section are orthogonal to each other and can be combined, some of them are only

Table 2.2: Other assumptions and variants.

Symbol	Explanation
CVX	Each client function $F_k(.)$ is convex.
SCVX	Each client function $F_k(.)$ is μ-strongly convex.
BNCVX	Each client function has bounded nonconvexity with $\nabla^2 F_k(x) \succeq -l \cdot \mathbf{I}$.
SMO	Each client function $F_k(.)$ is L-smooth.
BLGV	The variance of stochastic gradients on local clients is bounded.
BLGN	The expected squared norm of any stochastic gradient is bounded.
LBG	Clients use the full batch of local samples to compute updates.
AC	All clients participate in each round.
Prox	Use proximal gradient steps on clients.

Table 2.3: Convergence rates.

Method	Non-IID	Other assumptions	Variant	Rate
Yu et al. [YYZ18]	$(G,0)$-BGD	SMO; BLGV; BLGN	AC	$\mathcal{O}(\frac{1}{\sqrt{KT}})$
Khaled et al. [KMR19]	(G-B)-BGD	SMO; CVX; BLGV	AC; LBG	$\mathcal{O}(\frac{K}{T}) + \mathcal{O}(\frac{1}{\sqrt{KT}})$
Li et al. [LHY+20]	$(G,0)$-BGD	SMO; SCVX; BLGV; BLGN	-	$\mathcal{O}(\frac{E}{T})$
Karimireddy et al. [KKM+20]	(G,B)-BGD	SMO; BLGV	-	$\mathcal{O}(\frac{T(1-S/K)}{TS})$
FedProx [LSZ+18]	$(0,B)$-BGD	SMO; BNCVX	Prox	$\mathcal{O}(\frac{1}{\sqrt{T}})$

relevant in certain cases. Our work will focus on how to reduce the energy footprint in FL and we provide a brief survey of the existing methods on this topic in the next section.

2.2 Federated Learning Energy Cost

2.2.1 Existing areas of study

Prevalent ML models have a very large number of parameters so training such models requires high computational capability and communication bandwidth as well as large device memory. These requirements are not necessarily verified in the context of federated learning as clients can operate with limited bandwidth or computation resources and can therefore have difficulties in joining the FL framework. Dropping devices or waiting for them affects the convergence time or the training accuracy. This has led to significant interest in reducing both the communication and computation cost. Recent studies have focused on three main ways of reducing the cost of FL.

Federated learning requires a high number of communication rounds between the server and clients in order to share the model updates during training, leading to a huge communication cost when the model is in large size. For each round, a huge quantity of data may be spread throughout the network, easily reaching the order of GigaBytes for some DL architectures. To reduce this cost, a first approach consists in reducing the size of the object sent from clients to the server. Most studies focused on that point which can be considered as the bottleneck in FL, first because clients usually have slower upload connection speed than download. Moreover, as the objects communicated are then averaged across a large number of clients, reducing the size of the gradients communicated will have less impact on the overall performance of the model than compressing the parameters of the model that are sent back from the server to the clients. Upstream compression can even help to improve the privacy of the training process as it reduces the amount of sensitive data transmitted from the clients to the server. For example, Konečný et al. [KMY+16] successfully perform lossy compression on the client-to-server exchanges, reducing the model size transmitted from the remote devices to the server. They proposed structured and sketched local updates, the former forcing the parameter matrix of each client to change to a pre-specified low rank matrix and the latter encoding the client-side weights in a compressed form before sending them to the server. Their experiments indicate a communication cost reduction up to two orders of magnitude. Gradient compression will be one of the avenue considered in the following PhD as explained in Section 2.2.2.

Another way consists in reducing the size of the model that is sent from the server to the clients. Some works studied the impact of combining both downstream and upstream compression. [SWMS20] [CKMT18]. Caldas et al. extend the work of Konečný et al., by sending a lossy compressed model to the client but also by introducing Federated Dropout where clients can train small subsets of the global model.

A third lever to reduce the cost of FL consists of reducing the local computation cost for each client by making the local training procedure more efficient. However, in FL settings, communication costs dominate when compared to computational costs so increasing the computational costs and reducing the number of communication rounds is usually used to mitigate the overall cost [MMR+16].

Compression schemes affect the model's performance so a trade-off has to be found. Furthermore, a more efficient training that requires less updates meaning less communication and computation rounds can obviously lead to a reduced cost. Among the different upload schemes and algorithms we detailed previously, many of them actually already considered the high cost issue and proposed ways of tackling them, either by increasing the local computations made between communication rounds or by including stragglers or other considerations so as to converge in less communication rounds. There are also works that consider including a shared toy dataset in order to converge more quickly. [ZLL+18][WWL+21]

In our work, we will focus on data summarization as a method to reduce the overall consumption of FL.

2.2.2 Gradient Compression

In federated learning, clients participating in joint training only need to exchange their own gradients information without sharing private data. To reduce the communication overhead and improve the overall efficiency of federated learning, gradient sparsification and gradient quantization have been recently proposed as an alternative to the full gradient, only communicating the components with absolute values larger than a threshold [JA18, LHM+18, CSZZ20, HKMM21, LZCW21, MW21, KMY+16]. Since compression is a lossy process, the gains in terms of communication complexity are usually achieved at the expense of a worse iteration complexity. Therefore, previous works have only focused on the compression ratio and on the convergence rate of the training phase. There is not a good understanding of how gradient compression techniques impact the total energy consumption, whether they are worth applying in general, and how to achieve an optimal power-performance tradeoff with them. The initial objective for this internship was to fill these gaps but this will be studied in the following PhD.

2.2.3 Data Summarization

Machine learning has been pushed to the forefront in recent years partly owing to the availability of large datasets. However, training on massive data incurs a prohibitive energy cost and does not necessarily yield a better quality model as large datasets often contains redundant or noisy data. Massive training datasets also pose other several challenges including larger experimental turn around times and difficulty in hyperparameter tuning as well as higher costs and more time for labeling. Data summarization is a potential solution to this problem [KWHCJ19]. It aims at finding a representative subset of manageable size out of a large dataset. The hope is that training on a succinct data summary could both alleviate the energy consumption and result in a more accurate and robust model. Data summarization has been studied for centralised deep learning and it could also be pertinent for Federated Learning for the early rounds of training in the case where each client has a significant amount of data.

In the context of federated learning, we plan to investigate the following questions:

- **What are the energy savings enabled by data summarization ?** It seems reasonable to expect that training on a succinct summary rather than on the entire dataset directly can substantially reduce the energy consumption of the training phase. It is not clear however whether the extra energy required for data summarization is offset by the energy saved during the learning task thanks to the reduction in the amount of training data. Using summaries could also increase the required number of communication rounds and therefore the communication cost. Our objective will be to investigate whether it enables to reduce the total energy spent, using both an empirical approach and a theoretical study.

- **How to compute a relevant data summary while preserving user data privacy?** In parallel submodular maximization algorithms for big data summarization, the questions are how to distribute the data among the machines, which algorithm should run on each machine, and how to merge the resulting solutions. The question is radically different in the context of federated learning, as user data should be kept private, implying that there should be no data exchange between the participants. In that context, we have m local datasets $V_1, V_2, \dots V_m$, and the goal is to extract an informative summary A_i^* from each local dataset V_i so as to maximize the utility $f\left(\bigcup_{i=1}^m A_i^*\right)$, while ensuring a fair contribution of each participant (that is, constraints of the form $a_i \leq |A_i^*| \leq b_i$). The main question we plan to investigate is how to achieve this without any data exchange. We shall also study under which condition on the local datasets performance guarantee can be provided.

There are many methods for extracting informative summaries from large datasets, among them submodular optimization techniques are widely used. They formulate the problem as that of selecting a subset A^* of the entire dataset V optimizing a utility function $f : 2^V \to \mathbb{R}_+$ that quantifies the "representativeness" of the selected set A^*. In many applications, the utility function f is submodular, which basically reflects the fact that the added value of any element from the dataset decreases as we include more data points to the summary. Exploiting this property, a natural optimization problem in this context is to find a summary A^* of size at most k that maximizes the utility, that is, $A^* = \text{argmax}_{|A|\leq k} f(A)$. Alternatively, one might seek for a subset A^* of data elements which achieves a target fraction of the utility provided by the full dataset, that is, A^* is the smallest subset (in cardinality) of V such that $f(A^*) \geq (1-\epsilon)f(V)$.

Since the seminal work of Nemhauser [NWF78], many approximation algorithms for maximizing submodular set functions have been proposed, including algorithms with more general constraints than those mentioned above, parallel algorithms and streaming algorithms (see Chapter 2 of [Mir17] for a recent survey).

In the following paragraphs, we will see methods to optimize submodular functions and existing works on that for centralized deep learning.

2.2.3.1 Submodular maximization and submodular cover

We are given a large dataset V of size n. The goal is to extract from V a subset $S \subset V$ of data points which are most representative according to some objective function $f : 2^V \to \mathbb{R}_+$. For each $S \subseteq V$, $f(S)$ quantifies the utility of S. A set function f is naturally associated a *discrete derivative*, also called *marginal gain*,

$$\Delta_f(e|S) = f\left(S \cup \{e\}\right) - f\left(S\right),$$

which quantifies the increase in utility obtained when adding $e \in V$ to S.

Definition 1. *A set function $f : 2^V \to \mathbb{R}_+$ is submodular if $\Delta_f(e|A) \geq \Delta_f(e|B)$ for every subsets $A \subseteq B \subseteq V$ and every data point $e \in V \setminus B$. Furthermore, f is monotone if and only if $f(A) \leq f(B)$ for every subsets $A \subseteq B \subseteq V$.*

Submodular functions naturally model notions of information, diversity, and coverage in many applications. For example, let $s_{i,j}$ be the similarity between two elements $i, j \in V$ (e.g., $s_{i,j} = e^{\|i-j\|/\sigma}$). Then, the following function is submodular (non-monotone):

$$f(S) = \sum_{i\in V}\sum_{j\in S} s_{i,j} - \lambda \sum_{i\in S}\sum_{j\in S} s_{i,j},$$

where $\lambda \in [0,1]$. The first term is the traditional sum-coverage metric, while the second one penalizes similarity within S. Note that the function $f(S) = \sum_{i\in V} \max_{j\in S} s_{i,j}$ is another example of submodular function (see Chapter 3 of [Mir17] for other examples).

We can distinguish two different data summarization problems of interest:

- **Submodular maximization** : the goal here is to find a summary S^* of size at most k that maximizes the utility, that is,

$$S^* = \text{argmax}_{|S|\leq k} f(S).$$

- **Submodular cover** : the goal is to find a subset S^* of data elements which achieves a target fraction of the utility provided by the full dataset, that is,

$$S^* = \text{argmin}\left\{|S| \;:\; S \subseteq V \text{ s.t. } f(S) \geq (1-\epsilon)f(V)\right\}.$$

These optimization problems are NP-hard for many classes of submodular functions [KG05][Fei98][Wei22]. However, a simple greedy algorithm proposed by Nemhauser in [NWF78] is known to be very effective (see Algorithm 5).

Algorithm 5 Greedy algorithm

```
1: procedure GREEDY
2:     $S \leftarrow \emptyset$
3:     while $|S| < k$ do
4:         $v \leftarrow \text{argmax}_{e \in V} \Delta_f(e|S)$
5:         $S \leftarrow S \cup \{v\}$
6:     end while
7:     return $S$
8: end procedure
```

Theorem 1. *For the submodular maximization problem of any non-negative and monotone submodular function f, the greedy heuristic produces a solution S^g of size k that achieves at least a constant factor $(1 - 1/e)$ of the optimal solution:*

$$f(S^g) \geq \left(1 - \frac{1}{e}\right) \max_{[S] \leq k} f(S).$$

For the submodular cover problem, the approximation ratio of the greedy heuristic is $1 + \log(\max_e f(e))$, that is

$$|S^g| \leq \left(1 + \log\left(\max_e f(e)\right)\right) |S^*|,$$

where S^* is the smallest subset (in cardinality) of V such that $f(S^*) \geq (1 - \epsilon) f(V)$.

Interestingly, [Mir17] proposes an accelerated version of the greedy algorithm called Stochastic-Greedy that scales to voluminous datasets.

2.2.3.2 Continuous approaches for submodular functions

Given a submodular optimization problem, it is natural to want to extend the function f to a continuous function in order to use techniques from continuous optimization. The multilinear and the Lovász extensions of f are interesting in that sense.

We saw that the greedy algorithm gives good approximation guarantee for submodular maximization under cardinality constraints. However, there are other types of constraints. The multilinear extension f^m of f has the property of being concave along any line $d \geq 0$. This function can be used for submodular maximization under matroid constraint using the Continuous Greedy Algorithm. [CCPV11][Sin16] The multilinear relaxation can also be used for submodular maximization with other types of constraints such as a constant number of knapsack constraints [KST13] or a combination of matroids and $O(1)$ knapsacks [CVZ14].

The Definition 1, also called the diminishing marginal returns property, describes non-increasing discrete derivatives. In this aspect, submodularity seems very similar to the notion of concavity for continuous functions. However, as stated before, the submodular maximization problem is NP-hard. In contrast, it is known that the unconstrained minimization of submodular functions can be computed in polynomial time [GLS81] which brings the notion of submodularity closer to that of convexity for continuous functions. In the case of minimization, instead of being a utility function, the submodular function f could be a measure of some type of incoherence in the sets. Its Lovász extension can be used in order to efficiently find a minimizer of f.

[Wei22][Moi16] However, adding simple constraints can make submodular minimization very hard and studies focus on finding approximate solutions. [SF11]

I have briefly studied continuous approaches for submodular functions. However, we did not use them in our work so I will not elaborate on them.

2.2.3.3 Data subset selection for DNN

In the context of centralised deep learning, several works have studied the use of data subset selection to make models more efficient. Among the different methods, many use loss or loss gradient values for the objective function while others use geometric properties of the feature space. The goal can also be different as some methods aim for representativity, others for diversity for example. This section aims to present some significant works and areas of study in data subset selection for deep neural networks.

Researchers studied the use of data subset selection for active learning, a learning protocol where the algorithm can request the labels of a set of points chosen from a pool of unlabelled points. The goal is to find effective ways to choose the set of points to label in order to maximize the accuracy while minimizing the labeling effort.

In the context of active learning, [WIB15] use submodular functions for data subset selection to filter out the data samples with low uncertainty about predictions. To do so, they transform the data-likelihood functions for the Naive Bayes and Nearest Neighbor classifiers respectively into Feature based and Facility Location functions, making the data subset selection problem a constrained submodular maximization problem. The authors empirically show that these functions work well for other classifiers, including deep neural networks.

Specifically considering CNN and active learning tasks, [SS18] present a geometry-based method where they select representative examples based on coreset (weighted subset) construction, assuming that data points close to each other in the feature space tend to have similar properties. The goal is to minimize an upper bound of the gap between subset loss and whole set loss, their problem being equivalent to the k-center problem which is a minimax facility location problem in the form of $\min_{s^{k+1},|s^{k+1}|\leq b} \max_i \min_{j\in s^k \cup s^{k+1}} d(x_i, x_j)$ where b is the max number of points to choose to label at round k and s^k the subset of points already labeled at that round. To solve the k-center problem, they use a greedy approach as initialization and introduce their own algorithm using a mixed integer program as a sub-routine. For the distance metric $d(\cdot,\cdot)$, they consider the l_2 distance between activations of the final fully-connected.

[KIK+19] optimize the Facility Location function modeling representation or the Minimum Dispersion function modeling diversity by a lazy greedy algorithm for hyperparameters tuning and active learning tasks. They highlight that the diversity function works better on active learning tasks on datasets where there is a lot of redundancy, especially on small datasets while the representation function has better performance on hyperparameters tuning. As computing the similarity function has a complexity of $O(|V|^2)$ where V is the whole dataset, they instead approximate it with a nearest neighbor graph as suggested in [WIB14].

[AZK+19] present BADGE, an uncertainty based method trying to find examples that are hard for the model to make inferences on. The goal is to select subsets that have a diverse and higher magnitude of hypothesized gradients to incorporate both predictive uncertainty and sample diversity into every selected batch. To do so, they take the label with the highest activation in the output layer as a hypothesized label and use the k-Means++ algorithm on the hypothesized gradient space of the samples. Also looking for data points that are near the decision boundary, [MVBA21] introduce Contrastive Active Learning where samples whose predictive likelihood diverges the most from their neighbors are selected to construct the coreset.

Leaving the active learning setting, [WJC+19] show that CNN training can be accelerated by randomly skipping mini-batches with 0.5 probability without hurting the accuracy.

Interestingly enough, [CYM+20] show that a small proxy model can be used to perform data selection. They perform the data selection using three different uncertainty-based methods. The first one uses the same method as in [SS18] but computing the feature representation from the proxy's final hidden layer instead of the whole model. They also use max entropy uncertainty sampling or keep the points with the higher forgetting events where the forgetting events are defined as the number of times an example is incorrectly classified after having been correctly classified earlier during training the model. This also show that the techniques developed for the active learning setting can be used more widely in deep learning.

[MBL20] introduce a method named CRAIG to select a weighted subset to approximate the gradient of the full dataset by maximizing a submodular function. The idea is to find a subset approximately converging to the solution that would have been obtained if training on the full dataset. The authors prove that the subset S that minimizes an upper-bound on the error of estimating the full gradient, maximizes a submodular facility location function and can be efficiently found using a fast greedy algorithm. Their results show a 3x speedup for training deep neural networks without losing significant performance. The authors also show that any incremental gradient method on the subset S, obtained by CRAIG, converges in the same number of epochs as it would on the full dataset V meaning that the induced speedup is inversely proportional to the size of S. [KSR+21] introduce GradMatch, a gradient matching method similar to CRAIG that minimizes an error term on the gradient difference between the selected weighted subset and the gradient of either the training or validation set. Their method uses a squared $L2$ regularization term over the weight vector w to discourage assigning large weights to individual samples. Their objective function being weakly submodular, instead of a simple greedy algorithm, they introduce an Orthogonal Matching Pursuit algorithm. The GradMatch algorithm has been shown to provide better error bounds on the estimation of the gradient compared to CRAIG. However, when selecting small fractions of the whole dataset, CRAIG seems to provide better performance [GZB22]. The authors of [KSR+21] also presented GLISTER [KSR+20], a method based on a bi-level optimization problem that aims to maximize the log-likelihood of the validation set. The outer objective is to select a subset S that is a good representation of the full set, while the inner objective is to optimize the model parameters θ with respect to S. GLISTER can get higher performance early in training than CRAIG and random uniform sampling but requires extra computational need as well as access to the validation data which is not always the case in Federated Learning.

Without talking about subset selection, submodular functions have also been used for batch selection as done in [JRSB19]. The authors develop a submodular sampling strategy for mini-batch SGD which improves models' generalization performance. To do so, they combine four different functions such that the obtained objective function is submodular. For a given point, these functions are : the uncertainty score which is the entropy of the current model, the redundancy score which is the minimum distance between the considered point and other points in the subset, the mean closeness score which is the distance to the mean of all examples, the feature match score which encourages diversity across each dimension in the feature space. Given the high sampling rate required, the authors in [JRSB19] decided to use Lazier than Lazy Greedy algorithm [MBK$^+$15] on different partitions in a divide-and-conquer strategy.

The works studied in this section have different goals and applications but all of them are data summarization methods for centralized deep learning. Our goal is now to find a way to apply data summarization for federated learning.

Chapter 3

Data Summarization Implementation

The review of the literature has allowed us to highlight the extent of the production of articles on Federated Learning over the last few years. Since the introduction of FedAvg in [MMR$^+$16], many methods were proposed to improve its convergence or address specific FL challenges. Indeed, unlike in centralized or distributed learning, the algorithms must be robust to statistical and systems heterogeneity. This makes the use of multi-model and personalization approaches necessary for certain problems. Many improvements to the federated averaging algorithm were also introduced to tackle the heterogeneity challenge and alleviate its ill-effects. However, training complex FL models requires a high number of local update rounds as well as communication rounds therefore leading to a high energy footprint. To reduce the overall consumption, we propose to study data summarization with the use of submodular functions applied to federated learning.

3.1 How to compute a data summary in the FL setup

Data summarization has already been studied in distributed settings, for example with distributed submodular maximization [Mir17]. However, this can't be applied in the context of federated learning where each client's data should be kept private. Moreover, keeping in mind the high communication cost of FL, data summarization should not lead to more communication rounds as this could easily offset the savings made by using less data. Considering that, data summarization techniques can be applied two ways for federated learning : either simply by doing subset selection locally for each client, or by making data summarization from the server. Performing data summaries from the server offers a number of advantages. Indeed, the server collects data from all the clients and therefore has an overview over all the clients that each client individually cannot have. This could enable it to detect redundancies between different clients, for example. Moreover, in many FL setups, the server has high computational capability compared to the clients. So how could we compute a data summary from the server ? Among the different avenues, the obvious idea would be to use data summarization techniques to reduce the clients local datasets. However, this would require that each client communicates to the server information on its data which can violate the privacy requirements and can very easily be too expensive. For example, for grad-matching based methods such as CRAIG or GradMatch, the server would need the per-sample gradient of the loss for each client's dataset. This can't be implemented in practical scenarios. Another possibility would be to apply data summarization techniques to choose the clients involved in the current FL round. This would require less data to be communicated between the clients and the server, however the server would need all the available clients to send information at the beginning of the training and possibly each R rounds in order to be able to choose the clients efficiently, therefore leading to more communication rounds. Another avenue would be to use data summarization techniques for gradient compression, for example by selecting for each client only a subset of the model's parameters to optimize, so as to reduce at the same time the local computations requirements and the size of the parameters object that is sent from the clients to the server. This could be interesting for training very large models.

In our work, we decided to focus on local data subset selection. As we saw earlier, data subset selection can be used in the centralized setup for active learning or for efficient hyperparameter tuning. In addition to those two applications, local data subset selection could also speed up the FL training for other reasons. Indeed, dropping or waiting for clients with relatively low resource computational environments can negatively affect the training in FL so training on a subset could improve the training while enabling significant reduction in the energy consumption. Moreover, in the FL setup, the different devices may be available only at specific periods in the day so being able to perform rounds of training more frequently could also help the training. For example, some devices may only be available at night, or when they are charging up. Reducing the local computation time may allow more rounds to be completed in a single night, in order to reach more of the devices affected by this constraint, to reach them several times, or simply to maximize the responses of these devices (in the event that some of them don't have the time to complete the computation in a single night - see previous point about stragglers). If the total number of rounds required does not increase, this would improve the turnaround time and the energy consumption. Meanwhile, this method also guarantees that confidentiality is maintained, ensuring that there is no exchange of data between participants.

3.2 Data summarization method

It is not the aim of this section to compare the different methods mentioned above in order to choose the best possible one. This would be a complex task, especially since some methods are better suited to particular tasks, e.g. diversity-oriented methods work better on redundant datasets. Our study aims to remain general and not data-specific. An important aspect in our choice of method is the presence of theoretical guarantees. Indeed, we're aiming to couple data summarization with Federated Learning, which means that a method that works well experimentally in centralized mode won't be guaranteed to work well in federated mode. Our main objective is to reduce the amount of data used for training without hurting the accuracy. To do so, gradient matching methods seem to be well suited. Indeed, methods such as CRAIG [MBL20] or GradMatch [KSR+21] come with error bounds on the subset gradient which could help us in our convergence analysis. Given that the models parameters are updated based on those gradients, by finding an optimal coreset (weighted subset) that approximates the full dataset gradients, we ensure that the parameters updates are approximately the same as if the entire datasets had been used while reducing the computation costs.

3.3 Problem Formulation

Let the initial FL problem be

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \text{ where } F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w). \tag{3.1}$$

with $f_i(w) = \ell(x_i; y_i, w)$ the loss for point $(x_i; y_i)$ and model parameter w. We have K clients and each client k has a local data set P_k with $n_k = |P_k|$.

As explained before, we decide to perform a local data summarization for each client. This guarantees data confidentiality and, on the other hand, we can also expect a reduction in computing costs. Each time the server sends the global parameters of model w to client k, client k fetches a subset S_k of data on which it will then locally perform the training for E epochs before sending the updated weights back to the server, which will then be able to update the model's global parameters according to what it has received from each client.

To link data summarization to FL, we'll look at $\nabla_w F_k(w)$. In FedSGD, each client sends $\nabla_w F_k(w)$ to the server that will update the weights accordingly. In FedAvg and most other approaches, each client updates local weights at each iteration, based on the $\nabla_w f_i(w)$ of the batch concerned. If on our S_k subset, we have a $\nabla_w F_{S_k}(w)$ which is a good approximation of $\nabla_w F_k(w)$ and which will therefore lead to a fairly similar updating of weights, we can potentially hope to obtain convergence guarantees fairly close to those without data summarization. One avenue to explore would be to couple this with FedProx, to prevent the local model from moving too far away from the global model, especially if the S_k chosen after receiving the global weights is no longer relevant a few epochs later.

In order to choose S_k, the first idea is to find a subset on which the mean of the gradients used to update the weights is the same. This corresponds to minimizing :

$$\Big|\Big| \frac{1}{|P_k|} \sum_{i \in P_k} \nabla_w f_i(w) - \frac{1}{|S_k|} \sum_{i \in S_k} \nabla_w f_i(w) \Big|\Big| \tag{3.2}$$

In the case of FedSGD, updating the weights at iteration $t+1$ for client k and with a learning rate η corresponds to

$$w_{t+1}^k \leftarrow w_t - \eta \cdot \frac{1}{|P_k|} \sum_{i \in P_k} \nabla_w f_i(w).$$

With S_k found by minimizing (3.2), the update would be

$$w_{t+1}^k \leftarrow w_t - \eta \cdot \frac{1}{|S_k|} \sum_{i \in S_k} \nabla_w f_i(w).$$

The problem in (3.2) is not submodular. Another way of seeing it would be to use the Exemplar Based Clustering case which can be transformed into a monotone submodular function [Mir17] by minimizing:

$$\frac{1}{|P_k|} \sum_{i \in P_k} \min_{j \in S_k} \Big|\Big| \nabla_w f_i(w) - \nabla_w f_j(w) \Big|\Big| \tag{3.3}$$

With $\sigma(i) = \text{argmin}_{j \in S_k} ||\nabla_w f_i(w) - \nabla_w f_j(w)||$, $A_j = \{i \in P_k, \sigma(i) = j\}$ and $\alpha_j = |A_j|$, we then have:

$$\begin{aligned}\frac{1}{|P_k|}\sum_{i\in P_k}\min_{j\in S_k}\left|\left|\nabla_w f_i(w)-\nabla_w f_j(w)\right|\right| &= \frac{1}{|P_k|}\sum_{i\in P_k}\left|\left|\nabla_w f_i(w)-\nabla_w f_{\sigma(i)}(w)\right|\right| \\ &\geq \frac{1}{|P_k|}\left|\left|\sum_{i\in P_k}(\nabla_w f_i(w)-\nabla_w f_{\sigma(i)}(w))\right|\right| \\ &\geq \frac{1}{|P_k|}\left|\left|\sum_{i\in P_k}\nabla_w f_i(w)-\sum_{j\in S_k}\alpha_j\nabla_w f_j(w)\right|\right|\end{aligned}$$

After minimizing (3.3), we can choose this weight update:

$$w_{t+1}^k \leftarrow w_t - \eta\cdot\frac{1}{|P_k|}\sum_{j\in S_k}\alpha_j\nabla_w f_j(w).$$

Let L_k be the function such that $L_k(S)=\frac{1}{|P_k|}\sum_{i\in P_k}\min_{j\in S}||\nabla_w f_i(w)-\nabla_w f_j(w)||$. We therefore have $L_k(P_k)=0$. As in Baharan Mirzasoleiman's thesis, we set g_k such that $g_k(S)=L_k(\{e_0\})-L_k(S\cup\{e_0\})$ with e_0 a phantom point chosen such that $\max_{i'\in P_k}||\nabla_w f_i(w)-\nabla_w f_{i'}(w)||\leq||\nabla_w f_i(w)-\nabla_w f_{e_0}(w)||$, $\forall i\in P_k$. We then have g_k monotone submodular.

In order to guarantee convergence, we first decided to pose the problem as a submodular cover problem, i.e. for a given $\epsilon_k\in\,]0,1[$, we'll look for the smallest possible subset $S_k\subseteq P_k$ such that $g_k(S_k)\geq(1-\epsilon_k)g_k(P_k)$. Note $F_{S_k}(w)=\frac{1}{|P_k|}\sum_{j\in S_k}\alpha_j f_j(w)$. Thus:

$L_k(\{e_0\})-L_k(S_k\cup\{e_0\})\geq(1-\epsilon_k)\cdot[L_k(\{e_0\})-L_k(P_k\cup\{e_0\})]$

$L_k(\{e_0\})-L_k(S_k)\geq(1-\epsilon_k)\cdot[L_k(\{e_0\})-L_k(P_k)]$ thanks to our choise of e_0

$L_k(S_k)\leq\epsilon_k L_k(\{e_0\}$ because $L_k(P_k)=0$

yet $\frac{1}{|P_k|}\left|\left|\sum_{i\in P_k}\nabla_w f_i(w)-\sum_{j\in S_k}\alpha_j\nabla_w f_j(w)\right|\right|\leq\frac{1}{|P_k|}\sum_{i\in P_k}\min_{j\in S_k}\left|\left|\nabla_w f_i(w)-\nabla_w f_j(w)\right|\right|$

i.e. $\left|\left|\nabla F_k(w)-\nabla F_{S_k}(w)\right|\right|\leq L_k(S_k)$

so $\left|\left|\nabla F_k(w)-\nabla F_{S_k}(w)\right|\right|\leq\epsilon_k L_k(\{e_0\})$

Let $\sigma_k=\epsilon_k L_k(\{e_0\})$.

Let S^* be the smallest subset of P_k verifying the previous inequality. The greedy algorithm allows us to find a S_k^g such that $|S_k^g|\leq(1+\log(\max_e f(e)))\,|S^*|$.

With this setup for the data summarization, we adapted the convergence proof of FedAvg from [LHY+20]. This can be found in Appendix A.

However, this method requires to find e_0 satisfying $\max_{i'\in P_k}||\nabla_w f_i(w)-\nabla_w f_{i'}(w)||\leq||\nabla_w f_i(w)-\nabla_w f_{e_0}(w)||$, $\forall i\in P_k$ and that minimize $\sum_{i\in P_k}\left|\left|\nabla_w f_i(w)-\nabla_w f_{e_0}(w)\right|\right|$ under the previous constraint. This is in itself another optimization problem. Other works usually take the 0 point as their auxiliary element [Mir17], however the theoretic results obtained with the optimal e_0 can't be verified in that case.

Another way of transforming the problem in (3.2) into a monotone submodular one is to put it in the Facility Location form and to maximize:

$$\sum_{i\in P_k}\max_{j\in S_k}s(i,j) \tag{3.4}$$

where $s(i, j)$ is a similarity measure between points i and j. To do so, we define $d(i, j) = \left|\left|\nabla_w f_i(w) - \nabla_w f_j(w)\right|\right|$ and $s_k(i, j) = \max_{e,e' \in P_k} d(e, e') - d(i, j)$.

This formulation is very close to the problem addressed in [MBL20], and it becomes the same for their application to DNNs. It also enables us to use the greedy algorithm and its accelerated versions. In the following, we will refer to Problem (3.4) and the problem addressed by [MBL20] in the same way (mainly as CRAIG problem).

We focused on this section on submodular functions and did not present equations on [KSR+21] method or other gradient matching methods as we did not use them in our experimentations, as explained in the next section.

3.4 Improving the data subset selection step

Before connecting the data subset selection step with Federated Learning, we first tried to make that step the most efficient possible. In this section, we focus on subset selection in a centralized setup.

3.4.1 Last-layer approximation.

Computing the exact gradient of the loss is computationally expensive. In addition to that, as the number of parameters in modern DNN can be very large, computing the pairwise similarity for the greedy algorithms (or the orthogonal matching pursuit algorithm for GradMatch) on those very high-dimensional gradients comes with a prohibitive computation cost and slowed-down turnaround time which can even offset the advantages of training on smaller sets. To improve the speed and reduce the computation cost, last layer approximations can be used instead of the exact gradients [MBL20][AZK+19][KSR+20][KSR+21]. We trained small CNN models on MNIST data and very small subsets of only 50 images chosen out of a training dataset of 30000 images and compared using the last-layer approximation as implemented in [MBL20] with using the exact gradient. In our example, the model had a total of 62006 parameters so the exact gradient had a size of 62006 while the approximation had a size of 10 corresponding to the number of classes in our classification task. Using the approximation resulted in a reduction of 35% of the runtime for extracting a subset of 50 images out of 30000 and a drop of 2% of test accuracy for 5 epochs of training (0.73 with the approximation and 0.75 without). This time reduction would be larger as the models and datasets get larger and more complex.

3.4.2 Python packages for submodular optimization.

Using an efficient subset selection algorithm is critical to good performance on our experiments. Existing open-source Python libraries implement submodular maximization or directly subset selection. Both submodlib [KRI22] and apricot [SBN20] are Python libraries for submodular optimization. If submodlib uses a C++ optimization engine, the apricot software is fully implemented in Python using both numpy and numba to accelerate computation. We expect those libraries to be slower than commercial tools such as the smr.ai tool, which is implemented in C++ and is the one used for the results of the paper [MBL20]. The authors of [MBL20] also provided an open-source code implementing the Facility Location problem using the heapq package for their greedy algorithm. We compared it in terms of runtime with apricot and submodlib on the Facility Location function. On small datasets of size

configuration	50/500	500/5000	500/20000	5000/20000
submodlib	0.15	2.7	46	62
apricot	1.6	2.0	8	12
craig code	0.15	3.0	35	> 160

Table 3.1: Average runtimes in seconds for the different Facility Location implementations. A configuration S/V refers to the selection of a subset of size S out of a whole set of size V, each point in the dataset being of dimension 10.

5000, the three implementations can extract a subset of 500 elements in 2 or 3 seconds. On very small datasets of size 500, apricot is less efficient than the other two to extract a subset of 50 elements. However, on bigger datasets (20000), apricot seems to be far more efficient to extract a subset of 500 elements or 5000 elements. Table 3.1 shows average runtime values obtained on ten different datasets for 4 configurations. We note those configurations S/V where S is the size of the subset extracted out of a whole set of size V. For all configurations, the points in the datasets are of dimension 10.

3.4.3 Python packages for subset selection.

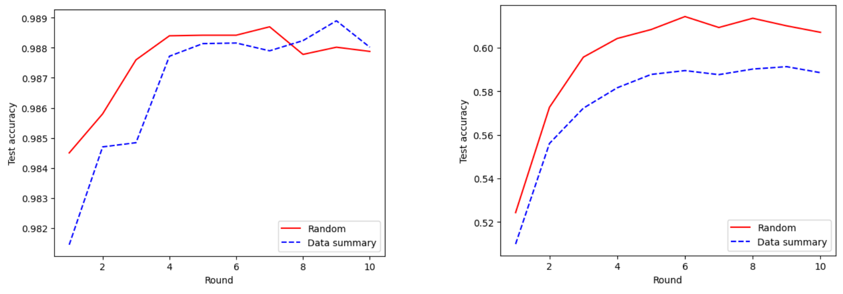
DeepCore [GZB22] and cords Python packages directly implement subset selection methods. The DeepCore package was created in order to fairly compare different methods. However, its implementation of methods such as CRAIG is slower in terms of runtime than what we could do using apricot or the code provided in the original paper. The package cords also implements several state-of-the-art data subset selection algorithms, including CRAIG and GradMatch. If their implementation of CRAIG seems slightly slower than the one of the original paper, their implementations of GradMatch or Glister are really fast in terms of runtime. However, the code provided is less easy to master due to a large number of undocumented parameters to be filled in. When combined with Federated Learning, using the dataloaders created from using cords results in some errors requiring modifications to the library code. It also easily creates dependency conflicts with other packages, therefore making it hard to use on different machines. For this reason, we preferred using the code provided by CRAIG and the apricot package for subset selection even if we believe that using GradMatch method could possibly lead to similar or better results.

3.4.4 Comparison with random uniform sampling

To assess the efficiency of a subset method, we can compare the performance with the one of training on the full dataset or training on random subsets. For the random sampling baseline, the subset is selected with random uniform probability out of the whole dataset. The paper introducing CRAIG [MBL20] presents results demonstrating that CRAIG can outperform random sampling when training DNN on MNIST and CIFAR10 datasets, leading to an observed speedup. These results were obtained using an industrial tool and not the open-source code provided with the paper. Nevertheless, [GZB22] shows that CRAIG and GradMatch almost never outperform random sampling on CIFAR10, but can outperform it on ImageNet for subset sizes inferior to 1% of the whole dataset size. Table 3.2 shows an extract from the table of their results on CIFAR10. However, [GZB22] only considered test accuracy but not the runtime or anything else that could be linked with energy gains. [KSR$^+$21] presents results considering the trade-off between speed-up (in terms of running time) and performance and showing that GradMatch outperforms the random sampling in

Fraction	0.1%	0.5%	1%	5%	10%	20%	30%	40%	50%	60%	90%	100%
Random	21.0 ± 0.3	30.8 ± 0.6	36.7 ± 1.7	64.5 ± 1.1	75.7 ± 2.0	87.1 ± 0.5	90.2 ± 0.3	92.1 ± 0.1	93.3 ± 0.2	94.0 ± 0.2	95.2 ± 0.1	95.6 ± 0.1
Craig	22.5 ± 1.2	27.0 ± 0.7	31.7 ± 1.1	45.2 ± 2.9	60.2 ± 4.4	79.6 ± 3.1	88.4 ± 0.5	90.8 ± 1.4	93.3 ± 0.6	94.2 ± 0.2	95.5 ± 0.1	95.6 ± 0.1
GradMatch	17.4 ± 1.7	25.6 ± 2.6	30.8 ± 1.0	47.2 ± 0.7	61.5 ± 2.4	79.9 ± 2.6	87.4 ± 2.0	90.4 ± 1.5	92.9 ± 0.6	93.2 ± 1.0	93.7 ± 0.5	95.6 ± 0.1

Table 3.2: Coreset selection performances on CIFAR10 from [GZB22].



(a) MNIST dataset (b) CIFAR10 dataset

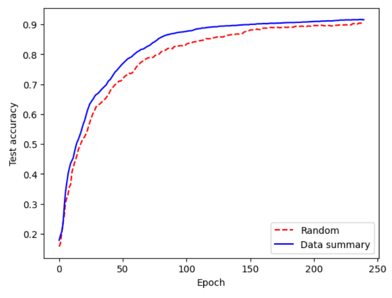
Figure 3.1: Test accuracy compared between random sampling and CRAIG sampling. One round corresponds to $E = 5$ epochs and the batch size is of 32. Full set size is 50000 and subset fraction size is 0.5. Centralized setup.

many cases where CRAIG does not. However, as said before, the implementation for CRAIG is the one from the library coords which is slower than the one used in [MBL20].

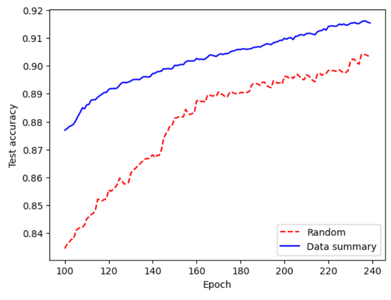
In our work, we first compared CRAIG method with random sampling. Those tests were done using an RTXA6000 GPU. The same CNN architecture was used, ten different initializations were randomly generated and the results correspond to the average performance on those ten models. Given n the size of the full dataset and k the size of the desired subset, the greedy complexity is $\mathcal{O}(nk)$ and stochastic greedy $\mathcal{O}(n)$. This results in a higher runtime and energy consumption for the CRAIG method as the dataset or subset sizes grow larger. We first compare CRAIG method and random sampling for selecting half of the original dataset. As shown in Figure 3.1, the random sampling method actually outperforms the CRAIG method with our setup, especially on CIFAR10. The training runtime on MNIST data are of 14 minutes 39 for CRAIG and 2 minutes 14 for random, on CIFAR10 it is of 2 hours 35 minutes for CRAIG and 2 minutes 40 for random. Meanwhile, the results from table 3.2 show that randomly selecting half of the dataset only results in a 2% drop in accuracy with their setup. Considering that, we also decided to apply CRAIG to a random half of the full dataset in order to reduce the runtime and the energy consumption while keeping the advantages of gradient matching methods.

Figures 3.2 and 3.3 show the evolution of the test accuracy on two different configurations where the subset fraction sizes are smaller. Out of the 50000 images of the MNIST training set, a subset of 200 elements is generated every E epochs. MNIST being a simple dataset with images looking a lot alike, the assumption made here is that CRAIG could outperform random sampling for small subsets.

When considering only the epochs, the CRAIG method outperforms the random sampling. In the example showed in Figure 3.2, it takes an average of 444 epochs for the model to reach a test accuracy of 0.92 using random sampling while it takes 262 epochs for it with CRAIG data summaries. On closer inspection, the accuracy also varies more widely with random sampling. However, our implementation of CRAIG is very slow while the random sampling is immediate and needs almost no energy to prune the dataset, making it quite efficient to reduce the required

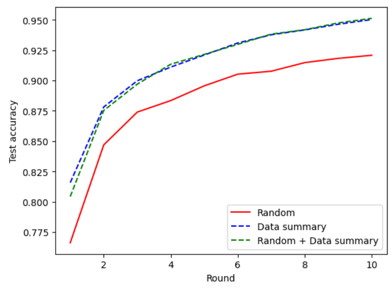


(a) Whole training

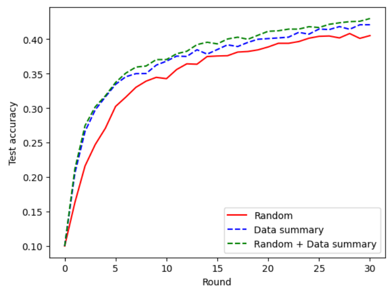


(b) Zoom on the end of training

Figure 3.2: Test accuracy on MNIST dataset compared between random sampling and CRAIG sampling. $E = 5$ and the batch size is of 64. Full set size is 50000 and subset fraction size is 0.004. Centralized setup.

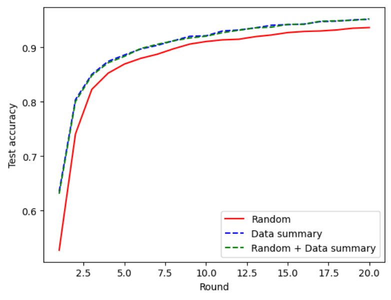


(a) MNIST dataset

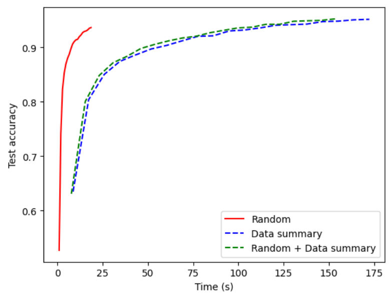


(b) CIFAR10 dataset

Figure 3.3: Test accuracy compared between random sampling, CRAIG sampling and combining both. One round corresponds to $E = 10$ epochs and the batch size is of 32. Full set size is 50000 and subset fraction size is 0.004. Centralized setup.



(a) Nb of rounds as x-axis.



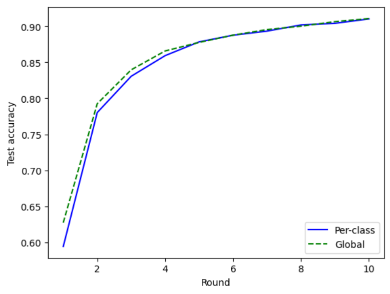
(b) Runtime as x-axis.

Figure 3.4: Test accuracy on MNIST dataset compared between random sampling, CRAIG sampling and combining both. One round corresponds to $E = 5$ epochs and the batch size is of 32. Full set size is 5000 and subset fraction size is 0.04. Centralized setup.

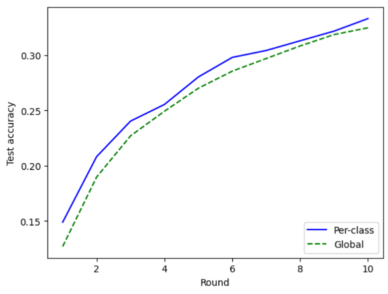
energy for training. In our example, reaching the 0.92 accuracy took 6 minutes with random sampling and 1hour and 13 minutes on average with CRAIG. We assume that better implementations, such as the one of the commercial tool used for the experimentations in [MBL20], could give better results and that the subset selection time could become negligible compared to the training time for larger models and more complex tasks than MNIST. We did not try to prove that as this is not the aim of our study. Of course, in the centralized setup and for such an easy task, using the full dataset largely outperforms any method using very small subsets. In only 5 epochs taking less than 1minute, the model is able to reach an accuracy of 0.98 fo MNIST and 0.57 for CIFAR10. Figure 3.3 shows another setup where the subsets are selected less frequently with $E = 10$. In that setup, the gap between the CRAIG method and the random sampling is larger, showing that CRAIG seems to provide more meaningful subsets. We also implemented a combination of both method, first randomly selecting 25000 samples out of the 50000 images of the training set before applying CRAIG on that half set. As explained before, the pairwise similarity complexity is $\mathcal{O}(n^2)$ and stochastic greedy is $\mathcal{O}(nk)$ with n being the size of the set considered for CRAIG and k the size of the desired subset. Halving that set size can therefore greatly speed up CRAIG. Figure 3.3 shows that this method does not hurt the performance and can even improve it. Meanwhile, it reduces runtime for 100 epochs from 17 minutes to 5 minutes on MNIST data, and from 22 minutes to 9 minutes on CIFAR10 data. Note that the model used here for CIFAR10 is a small CNN (two convolutional layers and one linear), the accuracy obtained at the end of the training could largely be improved by training state-of-the-art models (ResNet architectures for example) but we did not do that here as the overall performance of the model is not the main goal and smaller models are quicker to train.

Figure 3.4 shows the test accuracy results on MNIST data for another setup where the whole training set is a part of the original MNIST training set, containing only 5000 images. A subset of 200 images is then selected at each round out of those 5000 images. On such small subsets, the difference of runtime between random sampling and the use of submodular maximization is still obvious with the 20 rounds taking 19 seconds for the random sampling, 2minutes 30seconds for the combination of random and data summarization using submodular maximization and 2minutes 53seconds for simple submodular maximization.

It's difficult to draw any conclusions about the usefulness of gradient match-



(a) MNIST.



(b) CIFAR10.

Figure 3.5: Test accuracy compared between per-class data summary or global data summary for IID data with $E = 5$. Training set size is 5000 and subset fraction size is 0.04. Centralized setup.

ing methods in centralized setup from our tests, particularly in view of the results obtained with random sampling, which is much faster and energy efficient. The results obtained in [GZB22] also seem to support this, given the results provided with random sampling which are quite close to the other methods while runtime or energy savings are not considered in that paper. However, it is possible that gradient matching methods outperform random sampling on more costly and time-consuming trainings.

3.4.5 Other parameters

An easy way to improve both the accuracy and the energy savings of CRAIG and gradmatch methods is using per-class selection. Indeed, storing the per-sample gradients requires high memory for large datasets. As stated before, the stochastic greedy algorithm complexity also depends on the size of the dataset. For classification problems, separately selecting subsets from each class can therefore speed up the data subset selection step. This was done in [MBL20] where the performance was actually improved by doing so while maintaining the class ratios in the whole data. On our experiments on MNIST, the average test accuracy after 10 rounds was similar in both cases as shown in Figure 3.5 but the per-class selection reduced training runtime from 5 minutes 45 to 1 minute 06. For CIFAR, the average test accuracy was improved with the per-class selection and the runtime reduced from 7 minutes 35 to 2 minutes 24.

Of course, the frequency we make the data summaries is a parameter to tune in this setup. We could for example make one only once at the beginning of the training or make one at each communication round, especially since the data summaries obtained depend on the current model's weights in our setup. Increasing that rate could increase the performance as we would choose subsets that are especially relevant for the current epoch but this is costly. Some trade-off has to be found. On the other hand, selecting the subsets at each epoch is not costly for random sampling and much more efficient than keeping the subset for more epochs.

Another aspect of CRAIG is that it uses submodular maximisation so the desired size of the dataset has to be defined beforehand. A strategy that could help training would be to start the training with relatively small subsets as the first epochs don't need extreme precision and increase the size of the subset as the training progresses in order to achieve greater accuracy, better generalization and avoid overfitting.

Another option is to use submodular cover where we define how much we want the objective function to be close to its value if the whole dataset was selected. Given a dataset V, we are looking for the smallest subset $S \subseteq V$ such that $f(S) \geq (1-\epsilon)f(V)$. However using submodular cover is not always appropriate. In our case, we defined f as a facility location function :

$$f(S) = \sum_{i \in V} \max_{j \in S} \; d_{max} - d(i,j)$$

where $d(i,j) = ||\nabla_w f_i(w) - \nabla_w f_j(w)||$ and $d_{max} = \max_{e,e' \in V} d(e,e')$.

With submodular cover, we are therefore looking for some subset S that verifies

$$\sum_{i \in V} \max_{j \in S} \; d_{max} - d(i,j) \geq (1-\epsilon)|V| \cdot d_{max}$$

$$\sum_{i \in V} \max_{j \in S} \; - d(i,j) \geq -\epsilon \; |V| \cdot d_{max}$$

$$\sum_{i \in V} \min_{j \in S} \; d(i,j) \leq \epsilon \; |V| \cdot d_{max}$$

For a given ϵ, the case where d_{max} is very large compared to most $d(i,j)$ would result in finding a very small $S^\star$. For example, imagine an MNIST classification problem where your model is able to predict very well the number 1 but not so much the number 2. Now imagine the gradients for the images labeled 2 are quite large and close to each other while those for the number 1 are smaller and close to each other also. Given e labeled 1 and e' labeled 2 the two elements such that $d_{max} = d(e,e')$. If we have d_{max} which is very large compared to the distance between any image labeled 1 and e or the distance between any image labeled 2 and e', it is easy to picture how the algorithm could actually choose to select only two elements for the training. This is not a wanted behavior so the stop condition used for stochastic cover does not seem very appropriate in our case.

One idea to avoid this behaviour would be to change the d_{max} in the last equation $\sum_{i \in V} \min_{j \in S} d(i,j) \leq \epsilon \; |V| \cdot d_{max}$ by for example a d_{mean} defined as the mean value for the $d(i,j)$ where $i,j \in V$. To do so, we could redefine the problem as finding the smallest subset $S \subseteq V$ such that $f(S) \geq f(V) - \epsilon \; |V| \cdot d_{mean}$.

Another problem with submodular cover is the choice of the ϵ, especially since from one round to another, the variance of the distance matrix containing the $d(i,j)$ as well as the value for $f(V)$ might change depending on the training evolution.

Meanwhile, the advantage with submodular maximization is that by explicitely choosing the size, we can control the computation complexity of the algorithm. We still implemented submodular cover for stochastic and lazy greedy algorithms in the library apricot and think this is an avenue worth exploring.

Chapter 4

Federated Learning Implementation

After implementing subset selection for DNN, the next step is to implement the federated learning environment. For that, we used the Flower library.

4.1 Flower Package

Flower [BTM+20] is a Python package providing an easy-to-use infrastructure for federated learning. It is based on GRPC and is designed to work with large number of clients, up to several thousands. It is also compatible with a variety of ML frameworks like Pytorch, Keras or Tensorflow and it supports a wide range of devices and servers including Android, Nvidia Jetson, iOS and Raspberry Pi. The source code for Flower is available in the GitHub repository [flo23].

A federated learning environment consists on a server that executes the global computations and orchestrates the entire learning procedure, and a pool of available clients that have access to their individual data for training or evaluation of model parameters.

The architecture of the Flower core framework is depicted in Figure 4.1. The key elements are the training pipeline of clients, the RPC server for edge clients, the client manager and the *Strategy* abstraction. The *Strategy* abstraction represents the global logic for selecting clients, communicating to them as well as updating, aggregating and evaluating models on the server side. It corresponds to the *how* to proceed and contains information for example on which FL algorithm use (FedAvg, FedProx, etc). The *Client Manager*, also on the server-side, manages a collection of *Client Proxy* objects, each one representing a client connected to the server, based on the configured *Strategy.* It selects the clients and sends and receives messages to and from them. The server's architecture also relies on the FL loop, a bridge between the *Strategy* and the *Client Manager*, that orchestrates the entire learning process based on the information given by one or the other.

The client-side only waits for instructions given in the messages from the server. It uses accordingly its user-provided training or evaluation functions and sends back information to the server. Two types of clients can be used in the Flower framework. The edge clients live on real edge devices that communicate over RPC with the server. However, for the case where server and clients run on a single machine, Flower also provides special simulation capabilities through the Virtual Client Engine that creates *FlowerClient* instances only when they are actually necessary for training

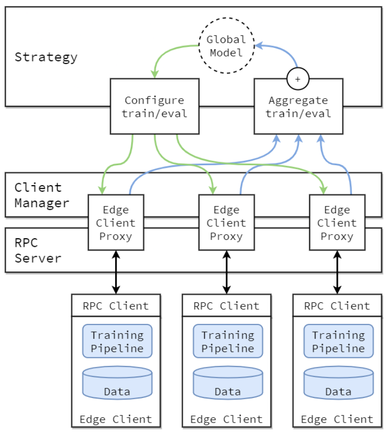


Figure 4.1: Flower core framework architecture with both Edge Client Engine and Virtual Client Engine. Edge clients live on real edge devices and communicate with the server over RPC. Virtual clients on the other hand consume close to zero resources when inactive and only load model and data into memory when the client is being selected for training or evaluation. [BTM+20]

or evaluation. This simulation mode allows a better management of the machine resources. We did not however use it, given that all clients participated to each training round in most of our experiments.

One advantage of the Flower framework is that it is highly customizable thanks to the *Strategy* and the *FlowerClient* instances.

4.2 Experiments

In figure legends, the terms ‘ds’ or ‘craig’ refer to the data summary obtained by solving Problem (3.4). The terms ‘r’ or ‘rd’ refer to random sampling. The term ‘full’ refers to using the full set.

We used a weighted variant of the cross-entropy loss and the Adam optimizer with its default parameters (except for the learning rate) for all clients in all our experiments.

For our tests, we will show the test accuracy as a function of the number of rounds or as a function of the runtime. Indeed, the communication cost depends on the number of rounds, while the computational cost depends on the running time.

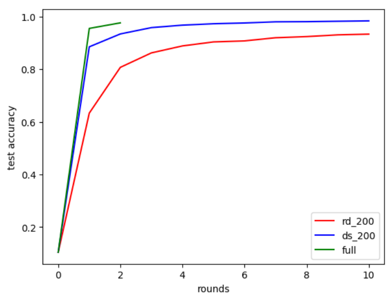
4.2.1 Results on IID data

Our first tests were done using Flower on a Quad Core Intel Core i7-7700 CPU. We first used MNIST dataset, splitting the training dataset into 10 datasets and splitting each of that dataset into a training and a validation set with a ratio of 90/10%. Therefore, each of our ten clients had a training set of 4500 images and a validation set of 500 images. We gave the MNIST test set to the server and used the implemented FedAvg [MMR+16] strategy of Flower.

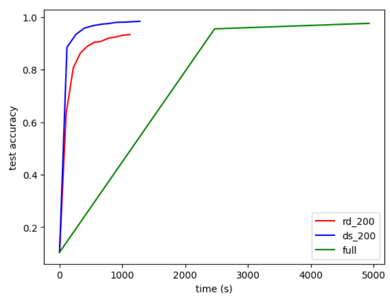
On MNIST, we compared extracting a data summary using the CRAIG method with random sampling or the full dataset, with the same initial model parameters. The results presented here are for the accuracy on the test set. For Figures 4.3 and 4.4, the subset size if of 500 images. It is of 200 for Figure 4.2. The FL training included $R = 10$ rounds of training where at each round the server sends the current global parameters to the ten clients, the clients locally trains the model for $E = 5$ epochs on the required dataset (full, random or data summary) and then sends its updated parameters to the server that aggregates all the responses to update the global model and so on.

We also ran some tests on CIFAR10 that are presented in Figure 4.5.

In the following figures, 'rd' refers to random sampling while 'ds' refers to data summarization using CRAIG method. The following number in the legend is the subset size.

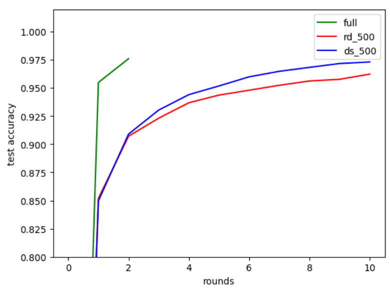


(a) Nb of epochs as x-axis.

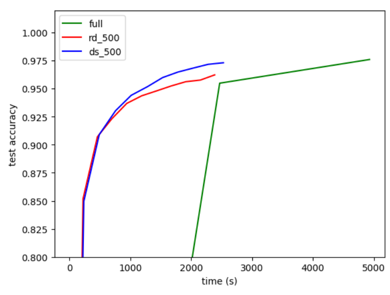


(b) Runtime as x-axis.

Figure 4.2: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for MNIST IID data. $E = 5$, $R = 10$, $\eta = 1e - 3$. Full set size is 50000 and subset fraction size is 0.0044. FL setup.



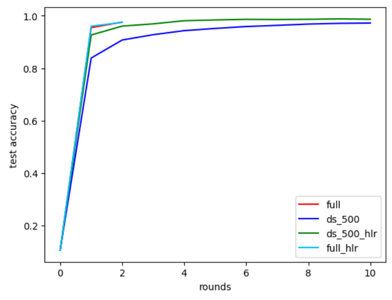
(a) Nb of epochs as x-axis.



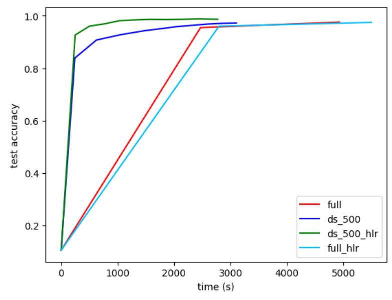
(b) Runtime as x-axis.

Figure 4.3: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for MNIST IID data. $E = 5$, $R = 10$, $\eta = 1e - 3$. Full set size is 50000 and subset fraction size is 0.011. FL setup.

Figure 4.2 shows that using subsets greatly accelerates training in terms of execution time when compared to the full dataset in our setup. However, this requires more rounds. In that configuration, we were able to reach a test accuracy of 0.980 in 8 rounds taking 18 minutes with the data summaries while it took 1h22 to complete 2 rounds of training on the full dataset to achieve a test accuracy of 0.976. In the FL setup, using submodular maximization seems this time to outperform random sampling. In 19 minutes corresponding to 10 rounds, the model was only able to reach an accuracy of 0.933 with random sampling. This comes from the fact that data selection time is here more negligible compared to the whole training process than what it was in the centralized setup. Here, as we select 200 images out of



(a) Nb of epochs as x-axis.

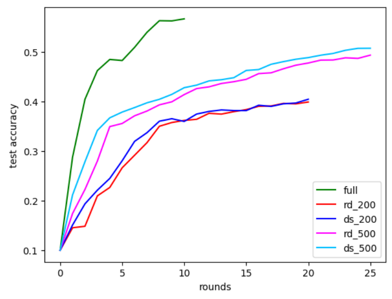


(b) Runtime as x-axis.

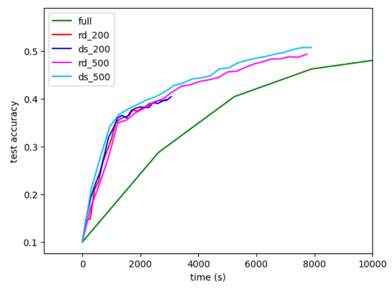
Figure 4.4: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for MNIST IID data. $E = 5$, $R = 10$, $\eta = 1e-2$ for the high learning rate (_hlr) and $\eta = 1e-3$ for the other. Full set size is 50000 and subset fraction size is 0.011. FL setup.

4500 images datasets for each client, the subset selection process is quite comparable to what we had in Figure 3.4, where training with random sampling took around 10 seconds and subdmodular maximization around 1 minute for 10 rounds. This 1 minute of difference is negligible for a 20 minutes training, especially when the subsets selected with the gradient matching technique lead to a better test accuracy.

Given that the communication rounds are very costly in FL, we tried to compensate the increased number of rounds by increasing the learning rate η. Figure 4.4 shows that increasing the learning rate did not really improve the training with the full dataset. For the data summaries however, increasing the learning rate in that configuration enabled the model to reach an accuracy of 0.96 in 2 rounds and 9 minutes when it took it 6 rounds and 35 minutes with the previous learning rate. In only 1 epoch, the model trained on the whole dataset was able to reach an accuracy of 0.954 in 41 minutes. Training on subsets therefore seems to greatly decrease the local training time but we were still not able on IID data to achieve a given accuracy in the same number of rounds as the full dataset training. This could however be possible on certain tasks or by adjusting some hyperparameters such as the learning rate η, the rate of subset selection (every E epochs in that configuration) or the subset size.



(a) Nb of epochs as x-axis.



(b) Runtime as x-axis.

Figure 4.5: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for CIFAR IID data. $E = 5$, $R = 10$, $\eta = 1e-3$. Full set size is 50000 and subset fraction size is either 0.004 or 0.01. FL setup.

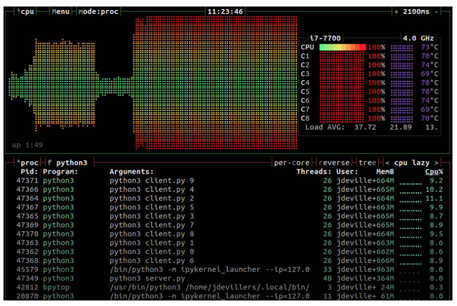
In the IID case, we would expect to obtain results quite similar to the centralized case, with notably faster convergence on the full set than on a subset, since all clients share a priori more or less the same local optimum, which is also the global optimum. Consequently, the results we had in this section are quite surprising given what we found in the centralized setup. Data subset selection seems far more relevant here, especially as the training takes more time. This shows that data subset selection, using random sampling or CRAIG sampling, can significantly speed up training for bigger models (with long training time) or for devices with limited system resources. This could help reduce the energy consumption in the case where reducing the dataset does not induce a drop in training accuracy and therefore the need for more computation rounds. Several hyperparameters are to be tuned for that, including the subset size, the number of local epochs and even the learning rate. The intuition on that is also that a smaller or even no drop in accuracy could be observed for non-IID data where the local models have different local optimum.

However the observed training times are not what we expected. The training on the full dataset split between the 10 clients takes more than 1 hour while it took seconds in the centralized setup. To be sure, we sequentially trained a model (same initialization) on the 10 different datasets and in a centralized setup. For one round, corresponding to 5 epochs of training on each dataset, the training took 50 seconds. As we suspect that the difference in time between the centralized setup and the federated may come from parallelization, we also run some tests changing the number of selected clients per round. We trained one model on MNIST dataset and using CRAIG sampling to select 200 images out of the 4500 images of each client. Table 4.1 shows the runtime in the sequential and the parallelized setup for one round of training, including the selection time and 5 epochs, and depending on the number of clients. For a small number of clients, the observed runtimes are actually better than for the sequential setup as the system seems to efficiently take advantage of parallel processing. For a bigger number of clients however, the runtimes are better when training sequentially. This probably comes from system limitations and this is worse when training on the full dataset. We contacted the Flower team on this matter and their answer regarding the full set training confirmed our doubts: ‘my best guess to explain the differences in time between 1 and 10 clients is that the latter takes much longer (much more than 10x longer) because each client (which in essence is a new process in your CPU) is fighting for the same resources as your other clients/processes. This translates in overheads’. Indeed, when displaying CPU usage during the training in the FL setup, we can see the clients ‘fighting’ for the resources as shown in Figure 4.6.

number of clients	sequentially (centralized)	concurrently (FL)
3	9 seconds	$4-6$ seconds
4	12 seconds	$5-7$ seconds
5	15 seconds	$7-9$ seconds
8	24 seconds	$28-31$ seconds
10	30 seconds	≈ 40 seconds

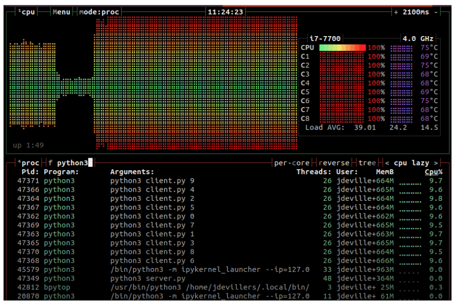
Table 4.1: Observed runtimes for one round of training depending on the number of clients. MNIST dataset and CRAIG sampling of 200 images out of 4500 for each client. $E = 5$.

This also shows how the runtime is not a perfect metric. All other things being equal, the runtime is a natural measure for efficiency. A common method to estimate energy consumption is to measure the training time and sample GPUs and



```
 ¹cpu   Menu  mode:proc            11:23:46                    + 2100ms -
                                                   i7-7700           4.0 GHz
                                                   CPU          100%    73°C
                                                   C1           100%    70°C
                                                   C2           100%    74°C
                                                   C3           100%    69°C
                                                   C4           100%    70°C
                                                   C5           100%    70°C
                                                   C6           100%    74°C
                                                   C7           100%    69°C
                                                   C8           100%    70°C
                                                    Load AVG:  37.72   21.89   13.
 up 1:49
 ⁴proc  f python3                            per-core  reverse  tree  < cpu lazy >
  Pid: Program:       Arguments:                       Threads: User:     MemB      Cpu%
 47371 python3        python3 client.py 9                    26 jdeville+664M ........  9.2
 47366 python3        python3 client.py 4                    26 jdeville+665M ........ 10.2
 47364 python3        python3 client.py 2                    26 jdeville+664M ........ 11.1
 47367 python3        python3 client.py 5                    26 jdeville+663M ........  9.9
 47365 python3        python3 client.py 3                    26 jdeville+665M ........  8.7
 47369 python3        python3 client.py 7                    26 jdeville+665M ........  8.9
 47370 python3        python3 client.py 8                    26 jdeville+664M ........  9.5
 47363 python3        python3 client.py 1                    26 jdeville+663M ........  8.6
 47362 python3        python3 client.py 0                    26 jdeville+662M ........  8.6
 47368 python3        python3 client.py 6                    26 jdeville+666M ........  8.9
 45579 python3        /bin/python3 -m ipykernel_launcher --ip=127.0  33 jdeville+963M .....  0.0
 47349 python3        python3 server.py                      48 jdeville+364M .....  0.0
 42812 bpytop         /usr/bin/python3 /home/jdevillers/.local/bin/   3 jdeville+ 24M .....  0.3
 20870 python3        /bin/python3 -m ipykernel_launcher --ip=127.0  11 jdeville+ 61M .....  0.0
```

(a)



```
 ¹cpu   Menu  mode:proc            11:24:23                    + 2100ms -
                                                   i7-7700           4.0 GHz
                                                   CPU          100%    75°C
                                                   C1           100%    69°C
                                                   C2           100%    75°C
                                                   C3           100%    68°C
                                                   C4           100%    68°C
                                                   C5           100%    69°C
                                                   C6           100%    75°C
                                                   C7           100%    68°C
                                                   C8           100%    68°C
                                                    Load AVG:  39.01   24.2    14.5
 up 1:49
 ⁴proc  f python3█                           per-core  reverse  tree  < cpu lazy >
  Pid: Program:       Arguments:                       Threads: User:     MemB      Cpu%
 47371 python3        python3 client.py 9                    26 jdeville+664M ........  9.7
 47366 python3        python3 client.py 4                    26 jdeville+665M ........  9.6
 47364 python3        python3 client.py 2                    26 jdeville+664M ........  9.8
 47367 python3        python3 client.py 5                    26 jdeville+664M ........  9.6
 47362 python3        python3 client.py 0                    26 jdeville+662M ........  9.6
 47369 python3        python3 client.py 7                    26 jdeville+665M ........  9.5
 47363 python3        python3 client.py 1                    26 jdeville+663M ........  9.7
 47365 python3        python3 client.py 3                    26 jdeville+665M ........  9.7
 47370 python3        python3 client.py 8                    26 jdeville+664M ........  9.5
 47368 python3        python3 client.py 6                    26 jdeville+666M ........  9.6
 45579 python3        /bin/python3 -m ipykernel_launcher --ip=127.0  33 jdeville+963M .....  0.0
 47349 python3        python3 server.py                      48 jdeville+364M .....  0.0
 42812 bpytop         /usr/bin/python3 /home/jdevillers/.local/bin/   3 jdeville+ 25M .....  0.3
 20870 python3        /bin/python3 -m ipykernel_launcher --ip=127.0  11 jdeville+ 61M .....  0.0
```

(b)

Figure 4.6: CPU usage for FL training with 10 clients at two different moments.

CPUs power consumption during training [QPFM+23][SGM19]. Indeed, the power consumption is likely to vary during the training or between different tests. Both the power consumption and the training time are highly hardware dependent with other factors coming into play such as the GPU, CPU and RAM technical specifications but also the use of parallelization as we saw before or other jobs running on the same machine.

To get around this problem, Floating Point Operations (FPOs) are another method for reporting efficiency, especially to compare deep learning models as it is hardware-agnostic and can be used to quantify the energy footprint of a model [SDSE19][DMPHO21]. FPOs (also called FLOPs) are the total number of floating point operations, additions and multiplications. However, fewer FPOs do not always result in faster processing [HHR+22], especially since it does not account for memory accesses and in our case, the communication cost would not be taken into account either.

4.2.2 First results on non-IID data

To simulate non-IID data, we distributed data corresponding to only 2 labels to each of our 10 clients. The trainset was therefore split into 10 sets, one for each label, and each client received half of two of those 10 sets.

Data subset selection could be used with any FL strategy, we first used FedAvg. However on non-IID data, the test accuracy tends to fluctuate a lot across the different rounds as shown in Figure 4.7.

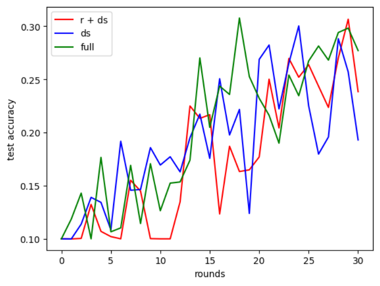


Figure 4.7: Test accuracy compared between random + data summary sampling, CRAIG sampling or the full dataset for CIFAR non-IID data using FedAvg. $E = 5$, $R = 30$, $\eta = 1e - 3$ Subset fraction size is 0.11. FL setup.

To avoid this fluctuation on this dataset, we can force the local updates to be closer to the current global model by using FedProx strategy instead of FedAvg. The objective function $F_k(w)$ of client k is modified by adding a proximal term $\frac{\mu}{2}\|w - w_t\|^2$ where w_t is the current global model, FedProx being equivalent to FedAvg when $\mu = 0$. Figure 4.8 shows the test accuracy when training on smaller datasets (training sets of 1000 images instead of the usual 4500) for different values of μ. With FedAvg ($\mu = 0$), the oscillations are really large but choosing a μ too large prevents the exploration of the parameter space. Out of the different values tested, $\mu = 0.01$ seems the optimal choice.

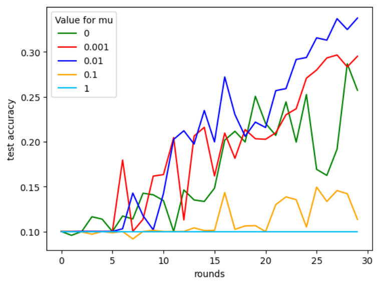


Figure 4.8: Test accuracy for CIFAR non-IID data using FedProx with different values for μ. $E = 1$, $R = 30$, $\eta = 1e-3$. FL setup. These experiments were done on smaller datasets for the sake of speed.

Despite the oscillations, the results in Figure 4.7 were quite promising, with all three curves following more or less the same trend. We repeat the same experiment with FedProx strategy. Figure 4.9 shows the results we obtained. We can see that the test accuracy increases very rapidly with the number of rounds in the full set configuration, but the other two strategies, CRAIG or random + CRAIG, catch up in the end. In terms of runtime as shown in Figure 4.10, there's no contest: the full set configuration takes much longer but, as we said earlier, this measure isn't very relevant here due to our resource limitations. It may, however, make sense for devices with no GPU and that are very CPU-limited. The results with a higher learning rate in Figure 4.11 show an acceleration of the convergence only for random sampling, that is able to reach a better accuracy than what we had in 50 rounds for the full set. This is quite promising for the random sampling, meaning that using random sampling could result in savings on the computation cost without necessarily requiring more rounds to achieve a given accuracy.

4.2.3 LAAS Computing Platform

At this point in our study, it becomes very complicated to draw conclusions, especially since our configuration is not really representative of a true federated system. To overcome this problem, the next step is to carry out the same experiments, but using different machines for the server and the clients. For this, we're using the LAAS HPC (High Performance Computing) cluster. It consists of different machines for a total of 316 cores, 3 GPUs (2 Nvidia RTXA6000 and 1 Nvidia A30) and 2TB of RAM. The whole is interconnected via 10 Gb ethernet links. The computing platform enables tasks to be launched via the SLURM resource manager, which executes user programs on the platform when the requested resources are available.

There are many advantages to using the computing platform. It allows us to carry out tests that will not be altered by parallel use on my own machine, thus

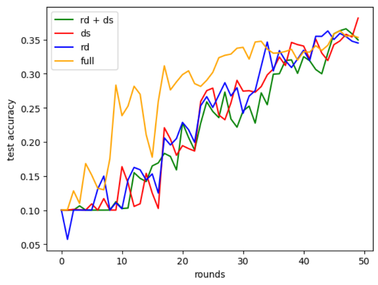
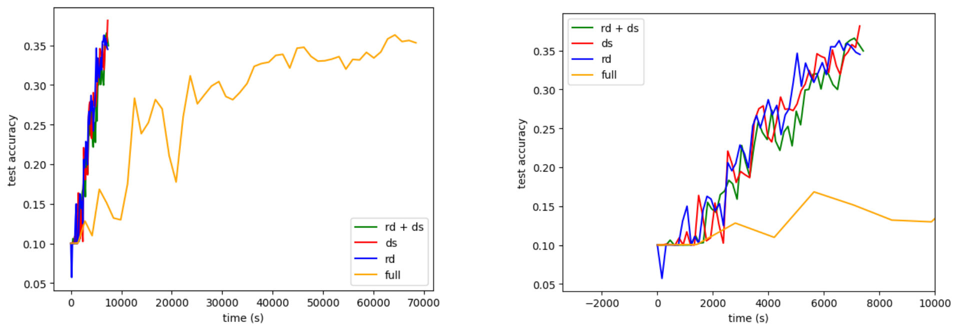


Figure 4.9: Test accuracy compared between random + data summary sampling (rd+ds), CRAIG sampling (ds), random (rd) or the full dataset for CIFAR non-IID data using FedProx. $E = 5$, $R = 30$, $\eta = 1e-3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup.



(a) (b)

Figure 4.10: Test accuracy compared between random + data summary sampling (rd+ds), CRAIG sampling (ds), random (rd) or the full dataset for CIFAR non-IID data using FedProx. $E = 5$, $R = 30$, $\eta = 1e-3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup. Runtime as x-axis.

guaranteeing a certain degree of reproducibility. What's more, we can use GPUs, which will greatly speed up testing. And obviously this provides a more relevant implementation of a federated configuration using different machines. However, this obviously comes with a few constraints, as the platform is accessible to all LAAS members, which means that we sometimes have to wait a while for machines to become available before tests can be launched. What's more, towards the end of the internship, the temporary storage space used to run the tests was full, preventing us from retrieving test logs, which was an unforeseen setback.

As using the computing platform greatly accelerates the training with the use of GPU, we also decided to use ResNet-18 as this model is the one commonly used in the other works for tests on CIFAR-10. [MBL20][QPFM+23][RCZ+21][KSR+21][KSR+20]

4.2.4 More relevant results on non-IID data

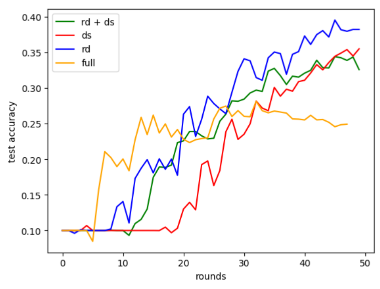
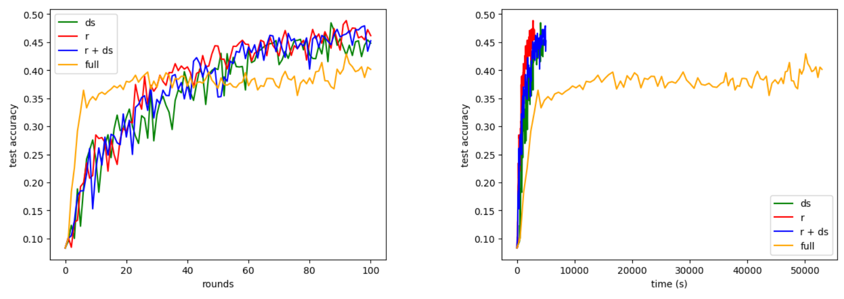


Figure 4.11: Test accuracy compared between random + data summary sampling (rd+ds), CRAIG sampling (ds), random (rd) or the full dataset for CIFAR non-IID data using FedProx. $E = 5$, $R = 30$, $\eta = 5e - 3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup.

4.2.4.1 Using CPU

In order to reproduce real-life conditions as closely as possible, these tests were carried out using 5 isolated machines (one for each client and one for the server) on the computing platform, on which no other tests were run at the same time. This configuration should solve the problem we had earlier with parallelization. For the clients, the different machines were : two Intel E5-2695 v3 2.3G, one AMD EPYC 7453 and one Intel 8352V 2.1G, each one having a minimum of 14 cores.



(a) Nb of rounds as x-axis. (b) Runtime as x-axis.

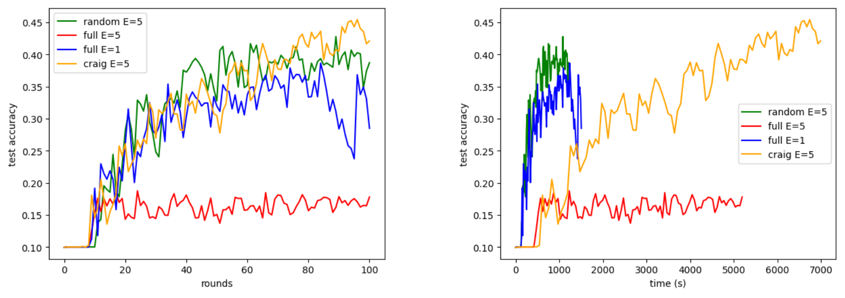
Figure 4.12: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for CIFAR-10 non-IID data, using ResNet-18. $R = 100$ for random and CRAIG, $R = 30$ for full. $\eta = 1e - 3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup with 4 clients on CPU.

The dataset was CIFAR10 and the model used was ResNet-18. The datasets were very non-iid with each client having labels that the others did not have. Unsurprisingly, using the full set takes too much time compared to using subsets, similarly to what we had in the previous experiments. The training with random sampling takes 51 minutes, $1h17$ with submodular maximization while using the full set takes almost 15 hours. In addition, the accuracy achieved with the full set is lower than that obtained on the subsets. This probably has to do with the learning rate or the number of local epochs.

The comparison between using submodular maximization (ds in the legend) or random sampling (r in the figure) tends to favor random sampling that takes less time and achieves the same or even better accuracy.

4.2.4.2 Using GPU

In this part we tested two configurations. In the first one, 10 Flower clients run on a single machine with an Intel 5218R 2.1G CPU (20 cores) and one GPU RTXA6000. The datasets are divided as in the previous section. In the second configuration, 4 Flower clients run on the same machine, 2 on one RTXA6000 GPU and 2 on another RTXA6000 GPU. The dataset is split so that each client has images corresponding to 5 labels : 1 to 5 for the first, 6 to 10 for the second, only the even-numbered for the third and the uneven for the last one. This second configuration is therefore less non-IID than the first one. Each local dataset contains 12500 images while they contain 5000 images in the first configuration. On both configurations, the server is run on another machine (Intel E5-2695 v3 2.3G CPU).



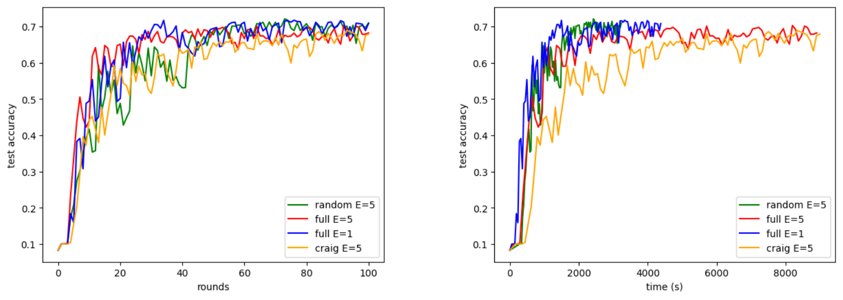
(a) Nb of rounds as x-axis. (b) Runtime as x-axis.

Figure 4.13: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for CIFAR-10 non-IID data, using ResNet-18. $R = 100$, $\eta = 1e{-}3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup with 10 clients on 1 GPU.

As shown in Figure 4.13, the configuration with 10 clients allows us to fall back on the results we had in the centralized configuration, where using CRAIG for data-summary is far too time-consuming compared with random sampling or even using the full set, as mentioned earlier in this report. However, random sampling seems to perform better than using the full set, even if we reduce the number of training epochs for each round to $E = 1$. This probably comes from the chosen data distribution. As we are in a non-iid configuration, each local model will be updated towards a local optimum which is not the same as the global one. If we let each client make a large number of local updates, their local model will be updated towards that local optimum which can be self-defeating for our training. This seems to be what happens in the case where $E = 5$ where the model fails to learn due to an excessive number of local updates.

In the configuration with 4 clients as shown in Figure 4.14, the conclusion for the CRAIG data summarization is even clearer. As each local dataset here contains 12500 images, with 1250 in the validation set and the rest in the training set, extracting a subset containing 1250 images (a 0.11 fraction as before) is even more time-consuming. Full set strategies perform better than random and CRAIG sampling in terms of test accuracy at each round, although random and CRAIG sampling soon catch up.

The results in both Figure 4.13 and 4.14 suggest that there are cases where using the full set even for only one epoch is not necessary, making the use of subsets



(a) Nb of rounds as x-axis. (b) Runtime as x-axis.

Figure 4.14: Test accuracy compared between random sampling, CRAIG sampling or the full dataset for CIFAR-10 non-IID data, using ResNet-18. $R = 100$, $\eta = 1e{-}3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup with 4 clients on 2 GPU.

relevant.

4.2.4.3 Frequency of subset selection

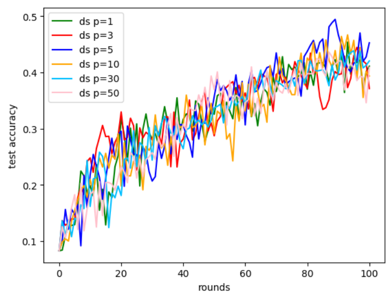
In the previous section, using CRAIG was too time-consuming. One way of limiting this is to play on the frequency with which the subset is selected. On all our previous experiments, we selected a new subset at each round but we can also decide to choose a new subset each p rounds. In this part, we use the same data partition as before with 10 clients, each having a training set of size 4500 representing 2 different labels for CIFAR-10. We distribute those 10 clients on 2 GPUs.

Figure 4.15 shows the test accuracy obtained for different values of the period p for random and CRAIG sampling. Interestingly enough, selecting the subset less frequently does not lead to a significant drop in precision. Meanwhile, this indeed allows us to save some computing time on CRAIG sampling.

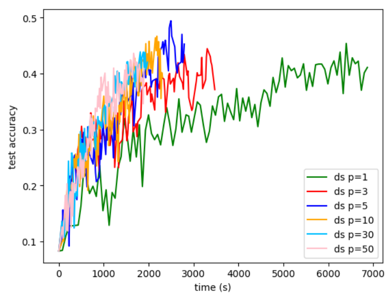
Figure 4.16 compares the test accuracy obtained using the full set with the one using random or CRAIG sampling each $p = 50$ rounds. When looking at the accuracy as a function of the number of rounds, the three methods produce close results. CRAIG sampling seems slightly better in this configuration. When looking at the time, random sampling is as expected the quickest but, for the first time in our experiments using GPU, the curve for CRAIG sampling closely follows that of random sampling while the one for using the full set is clearly below.

4.2.4.4 Size of subset

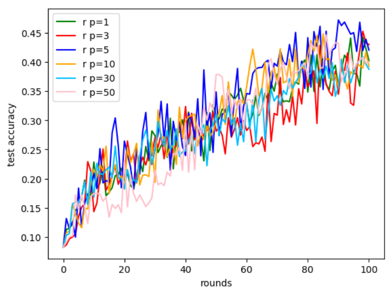
Of course, the size of the selected subset is a hyperparameter that can greatly impact the training. Figure 4.17 shows the test accuracy for different values of the subset size S for random sampling. If S is too large, the local model will move towards the local optimum which can be detrimental to the training. This is why using a subset of 2250 outperforms using the full set ($S = 4500$). A large value of S will also increase the runtime, especially when using submodular maximization for subset selection. If S is too small, the training will take much longer in terms of rounds which will result in an increased communication cost. This is what would happen



(a) Test accuracy per round for CRAIG sampling.



(b) Test accuracy as a function of time for CRAIG sampling.



(c) Test accuracy per round for random sampling.

Figure 4.15: Test accuracy for random sampling and CRAIG sampling for different values of the period p for extracting the subsets. CIFAR-10 non-IID data, using ResNet-18. $R = 100$, $\eta = 1e-3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup with 10 clients on 2 GPU.

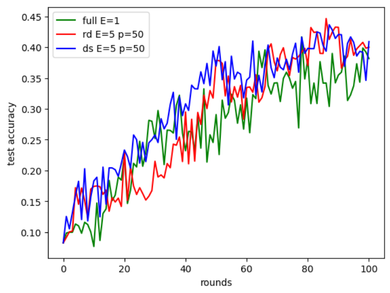
for $S = 500$ where the model learns and eventually achieves the accuracy reached by bigger subsets but this takes more rounds. A balance has to be found. In addition to that, the optimal value for S depends on the chosen value of E or even of the learning rate η.

4.2.5 Conclusion on experiments

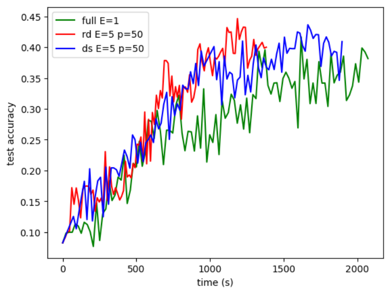
A summary of our different experiments and their results is provided in Appendix B. Based on the results of our experiments, we can clearly identify two cases in which the use of subsets of local datasets seems relevant.

The first case concerns clients with limited computational resources, such as in Section 4.2.2 or Section 4.2.4.1. In those cases, the training can take a very long time. The straggler problem is a real issue in federated learning and being able to include stragglers in the training loop can help the overall model performance, especially if the clients in question hold data that don't appear sufficiently in other datasets. Using subsets instead of the full set can significantly decrease the training time, particularly if those clients have large datasets, and therefore help them participate in the FL training.

The second case concerns clients with large datasets that are not representative of the whole data distribution. Basically, this means that their local optimum for the model's parameters is different from the global optimum. To prevent them from converging too much towards their local optimum, hyperparameters have to be chosen

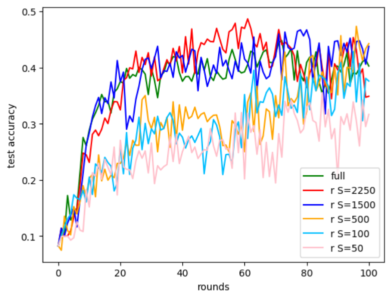


(a) Nb of rounds as x-axis.

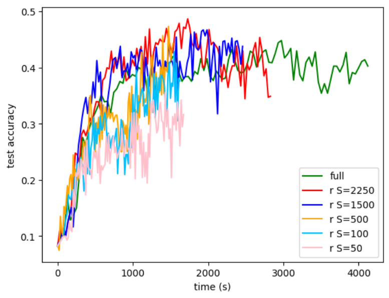


(b) Runtime as x-axis.

Figure 4.16: Test accuracy compared between full set or random sampling and CRAIG sampling with $p = 50$ for CIFAR-10 non-IID data, using ResNet-18. $R = 100$, $\eta = 1e-3$, $\mu = 0.01$. Subset fraction size is 0.11. FL setup with 10 clients on 2 GPU.



(a) Nb of rounds as x-axis.



(b) Runtime as x-axis.

Figure 4.17: Test accuracy compared for various sizes of the subset selected by random sampling for CIFAR-10 non-IID data, using ResNet-18. $R = 100$, $\eta = 1e-3$, $\mu = 0.01$. Subset fraction size varies. FL setup with 10 clients on 2 GPU.

wisely. This can be done for example by reducing the learning rate or increasing the batch size in order to reduce the number of local updates. However, this is not always the best choice for example if the training works better with the initial hyperparameters for other clients with smaller datasets and more generally if we can achieve the same performance simply by going through a smaller number of points of their datasets. This is our interpretation of what happens in Section 4.2.4.2 where using subsets leads to better results than using the full set for 1 epoch.

Selecting a subset also requires tuning some parameters including the size of the subset and the selection frequency. In our case, reducing the selection frequency did not lead to a drop in accuracy while, for the data summarization method, it helped reduce the running time as shown in Section 4.2.4.3. Considering the size of the subset, a balance has to be found as explained in part 4.2.4.4. Tuning those parameters can be done locally in order to avoid additional communication rounds. It could of course result in an additional computational cost but this could easily be offset for long and costly trainings, for example in a FL hyperparameter tuning context. However, it is important to point out that the use of subsets also influences the choice of certain hyperparameters, such as the number of epochs for local training in each round. More indirectly, the choice of other parameters, such as the learning rate, may depend on whether or not subsets are used.

The goal of this internship was also to study the use of submodular maximization for data summarization. Our approach focused on finding a subset approximating the full set gradient for the loss. Our study did not show clear improvements in the test accuracy or the loss. Considering the longer runtimes, this means that using submodular maximization rather than random sampling was not relevant in our case. This could be linked to the datasets we used or the way we distributed them between clients. Indeed, in our experiments, each client's local dataset would actually be quite homogeneous, with 5000 images of only 2 labels resulting in a lot of redundancy. Based on that, it is not surprising that random performed well. One might think that results would be different on more complex datasets, but this has not been studied here.

4.3 Quantifying the energy savings

It is difficult to estimate the energy cost of our experiments. While runtime is a good proxy for the computational cost, it does not capture the costs of communication in a real federated environment. Few works have studied the energy cost of federated learning but methods from deep learning can be applied to estimate the computational cost. Based on that, [QPB+21] and [QPFM+23] focus on the environmental cost of federated learning, considering both the training energy consumption and the networking energy.

To measure the computational cost, which highly depends on the client's hardware, they propose to sample the GPU and CPU power consumption at training time, as has been done for centralized deep learning [SGM19]. For each selected client, the training energy consumption for one round is the product of the wall clock time and the average power measured. This measure is not exact, especially since it does not take into account system memory and storage.

For the communication cost, they consider the energy consumed by routers and the energy consumed by the hardware when downloading and uploading the model parameters. For each selected client i and each round, this translates into an additional cost of $S \cdot (\frac{1}{D} + \frac{1}{U}) \cdot (e_r + e_{idle,i})$ where D and U are the download and upload speeds, S is the size of the model, e_r the power of the router and $e_{idle,i}$ the idle power consumption of hardware for client i.

Using this method, they provided a carbon footprint calculator[1] whose results are based on their experimental results. To use the calculator, the user has to fill in different parameters. They have to select the hardware profile of the devices between two choices : NVIDIA Tegra X2 and Jetson Xavier NX devices. These devices have been chosen as they can be found embedded in various IoT devices including cars, smartphones, and video game consoles. Then, a country has to be selected as this defines the conversion factor between energy and CO2 emissions as well as the upload and download speeds. The remaining parameters to define are the dataset used, the number of rounds, the number of local epochs and the number of selected devices at each round. As their dataset partition methodology is not at all similar to what has been done in this experiment, the results provided by the calculator can't directly be applied to our case. However, they only depend on the number of rounds and do not take into account a desired accuracy.

Just to have a rough idea of the potential energy savings, we decided to put in here different values given by their calculator. These values were obtained by

[1]available at `https://mlsys.cst.cam.ac.uk/carbon_fl/`

selecting France as the country, Nvidia NX as the device, 10 active devices at each round and 200 rounds. We also used the value of 0.054 kgCO2eq / kWh for the conversion factor as in their study. To obtain the results for using only half of the dataset (which is equivalent to what was done in our study when we used a subset of 500 images for 5 epochs instead of the full set of 5000 images for one epoch), we divided by 2 the difference between doing 1 or 2 local epochs and subtracted this value from that obtained for the full set (1 local epoch). We show results for both CIFAR10 and ImageNet, these were obtained using the ResNet-18 model. The parameters used here were chosen to illustrate as best as possible the experiment of Figure 4.13.

Table 4.2: Results from the FL carbon footprint calculator.

Setup	Footprint (gCO2eq)	Energy consumption (Wh)
CIFAR10 full dataset	1.81	33.52
CIFAR10 half dataset	1.73	31.94
ImageNet full dataset	117.87	2182.78
ImageNet half dataset	59.76	1106.67

The results in Table 4.2 show little difference for the CIFAR10 dataset where the communication cost accounts for the most part of the overall consumption. For ImageNet however, that has more and bigger images than CIFAR10, the communication cost is the same as for CIFAR10 as it is the same model used but the computation cost this time prevails. In that setup, reducing the size of the dataset by half leads to almost halving the overall energy consumption. This shows that for complex tasks where the computation cost is significant, reducing the dataset size can indeed result in substantial energy savings.

Chapter 5

Other avenues to explore

Other datasets ? Our study did not show any clear benefits to using submodularity as posed in our problem over random sampling. This is potentially linked to the simplicity of our datasets, especially when we consider that [GZB22] showed that CRAIG almost never outperformed random sampling on CIFAR10 but could outperform it on ImageNet for small subset sizes. This study could therefore be extended to other datasets. We can hope that the use of submodular maximization to produce subsets will be more relevant on more complex datasets and models. We also hope that the results concerning the relevance of using subsets remain valid, as do the potential energy savings mentioned in the previous section.

Other types of heterogeneity. Moreover, we only considered classes heterogeneity but did not explore the differences in the sizes of clients datasets. We could for example imagine a variant of FedNova where, instead of aggregating local models by number of local updates, we fixed that number so that each client performs the same quantity of computation. However, this presupposes a certain degree of homogeneity in the dataset distribution and the clients system resources. The first assumption is also a disadvantage of FedNova where the weight update of a client having only for example 3 images would contribute as much as another one to the global update. FedNova or the variant could be really useful when we have IID data but different sizes of datasets or when we have an imbalanced dataset with for example less images for a certain class that correspond to the dataset of a client. Without even mentioning FedNova, the use of subsets still seems appropriate for large datasets.

We talked about stragglers but in our experiments, all our clients had similar hardware and sizes of datasets. We could imagine configurations with totally different clients, where some could complete their training much more quickly than others, thus creating real stragglers. We could then imagine implementing data summarization but only for stragglers and see the impact of waiting for the stragglers, dropping them or using subsets.

Quantifying the degree and defining the type of non-iid. Federated learning inevitably comes with issues of heterogeneity, particularly with regard to data distribution. But as mentioned in the paragraph above, there is no single type of heterogeneity. In addition to the distribution heterogeneity, there is also the possibility that the dataset is imbalanced. In our study, clients had different labels proportions. Nevertheless, the union of our datasets was totally balanced. Solving the problem of skewed class proportions in a federated learning environment is another matter that needs to be addressed. Other types of heterogeneity can also exist, for example clients can have different optimizers.

[LHY+20] proposed $\Gamma = F^* - \sum_{k=1}^{K} p_k F_k^* \geq 0$ as a measure of the degree of non-iid but this value can't be accessed in practice. One that can be accessed is the inter-client gradient variance, $\mathbb{E}_k[||\nabla F_k(w) - \nabla f(w)||^2]$. However, this value is hard to interpret. Being able to quantify and define by how muche the data are non-iid could help better understand how to improve federated learning algorithms.

Complexifying our subset selection strategy. As explained in Section 3.4.5, we decided to use submodular maximization instead of submodular cover. However this means setting a desired size of the subset but as the model parameters converge, this desired size may become too small to achieve better accuracy. A good strategy could be to gradually increase the subset size.

Other definition of the subset selection problem Our idea was to select a subset on which the gradient of the loss should be similar to the one of the full set. However, this might not be the best way to select the subset and other methods could work better, for example by focusing on diversity or bad predictions, on balancing the full set or on eliminating noise.

Chapter 6

Conclusion

Statistical and systems heterogeneity is inherent to federated networks. Many algorithms already try to tackle this heterogeneity. In our work, we proposed to study the use of local subsets instead of clients full sets. We compared random sampling with the use of a gradient matching method based on submodular maximization. Our study did not show a clear interest in using the latter method rather than random sampling. However, we identified two cases where the use of subsets is relevant : for clients with limited computational resources or for clients with large datasets in non-iid setups. In the first case, training on the full set takes too much time and may lead to the server not including the client's updates. In the second case, training on the full set may lead to the client's local model to converge to its local optimum therefore moving away from the global optimum. In certain cases where the cost linked with computation prevails over the communication cost, reducing the local dataset size can help significantly cut the energy cost and therefore the carbon footprint of federated learning.

One point that recurs in most studies of federated learning, including our own, is that the results depend on the situation. It would therefore be interesting to be able to define in some way the degree of heterogeneity of a federated environment. Our study could also be extended to new datasets or different methods or strategies to select the subset.

Appendix

A Convergence proof of FedAvg adapted for data summarization

This proof corresponds to the convergence proof of FedAvg in [LHY+20], with a few adjustments for data summarization.

The assumptions in [LHY+20] are the following:

1. The F_k are all L-smooth.
2. The F_k are μ-strongly convex.
3. Let ξ_k be sampled from P_k uniformly at random, then $\mathbb{E}_\xi[||\nabla F_k(w_t^k, \xi_t^k) - \nabla F_k(w_t^k)||^2] \leq \sigma^2$.
4. $\mathbb{E}_\xi[||\nabla F_k(w_t^k, \xi_t^k)||^2] \leq G^2$.
5. All clients participate at each iteration.

For a given $\epsilon_k \in]0,1[$, the data summarization problem as explained in Section 3.3 is to find S^* the smallest subset verifying

$$g_k(S_k) \geq (1-\epsilon_k) g_k(P_k)$$

where

$$\begin{aligned}
&g_k(S) = L_k(\{e_0\}) - L_k(S \cup \{e_0\}) \text{ monotone submodular} \\
&\sigma_k = \epsilon_k L_k(\{e_0\}) \\
&L_k(S) = \frac{1}{|P_k|} \sum_{i \in P_k} \text{min}_{j \in S} ||\nabla_w f_i(w) - \nabla_w f_j(w)|| \\
&\sigma(i) = \text{argmin}_{j \in S_k} ||\nabla_w f_i(w) - \nabla_w f_j(w)|| \\
&A_j = \{i \in P_k, \sigma(i) = j\} \text{ and } \alpha_j = |A_j| \\
&F_{S_k}(w) = \frac{1}{|P_k|} \sum_{j \in S_k} \alpha_j f_j(w) \\
&\text{max}_{i' \in P_k} ||\nabla_w f_i(w) - \nabla_w f_{i'}(w)|| \leq ||\nabla_w f_i(w) - \nabla_w f_{e_0}(w)||, \forall i \in P_k
\end{aligned}$$

We also adapt the assumptions to our problem.

H.1 We assume that the F_k and the F_{S_k} are all L-smooth.

H.2 We also assume that the F_k and the F_{S_k} are μ-strongly convex.

H.3 In our case, we assume that we don't draw samples randomly and that our batches correspond to each local dataset in its entirety. Initially, we'll assume that data summarization is performed at each iteration. We can therefore use the S_k obtained by data summarization, which verifies $||\nabla F_k(w_{t_0}^k) - \nabla F_{S_k}(w_{t_0}^k)||^2 \leq \sigma_k^2$ for iteration t_0 where the data summarization has been done.

H.4 There is $G \geq 0$ such that $\mathbb{E}_k[||\nabla F_{S_k}(w)||^2] \leq G^2$.

H.5 All clients participate at each iteration. We then have for a variable X, $\mathbb{E}_k[X_k] = \sum_{i=1}^K p_k X_k$ with $\sum_{i=1}^K p_k = 1$ for example if $\forall k, p_k = \frac{n_k}{n}$.

Let w_t^k be the model parameter maintained in the k-th device at the t-th step. Let $\mathcal{I}_E$ be the set of global synchronization steps, i.e., $\mathcal{I}_E = nE | n = 1, 2, \cdot.$ If $t+1 \in \mathcal{I}_E$, i.e., the time step to communication, FedAvg activates all devices. Then the update of FedAvg with data summaries can be described as

$$v_{t+1}^k = w_t^k - \eta_t \nabla F_{S_k}(w_t^k), \tag{1}$$

$$w_{t+1}^k = \begin{cases} v_{t+1}^k & \text{if } t+1 \notin \mathcal{I}_E, \\ \sum_{i=1}^K p_k v_{t+1}^k & \text{if } t+1 \in \mathcal{I}_E. \end{cases} \tag{2}$$

Here, an additional variable v_{t+1}^k is introduced to represent the immediate result of one step SGD update from w_t^k. We interpret w_{t+1}^k as the parameter obtained after communication steps (if possible). In our analysis, we define two virtual sequences $\bar{v}_t = \sum_{k=1}^K p_k v_t^k$ and $\bar{w}_t = \sum_{i=1}^K p_k w_t^k$. For convenience, we define $\bar{g}_t = \sum_{i=1}^K p_k \nabla F_k(w_t^k)$ and $g_t = \sum_{i=1}^K p_k \nabla F_{S_k}(w_t^k)$.

Lemma 1. *Property of L-smooth and convex function If f is L-smooth and convex, we have*

$$f(z) \geq f(x) + \nabla f(x)^T (z - x) + \frac{1}{2L} ||\nabla f(z) - \nabla f(x)||^2, \forall x, z$$

Proof Assume f is L-smooth. Given x, we note $\phi_x(y) = f(y) - \nabla f(x)^T y$. We have $\forall y, z$

$$f(y) \leq f(z) + \nabla f(z)^T (y - z) + \frac{L}{2} ||y - z||^2$$
$$f(y) - \nabla f(x)^T y \leq f(z) - \nabla f(x)^T z + \nabla f(z)^T (y - z) + \frac{L}{2} ||y - z||^2 \nabla f(x)^T (z - y)$$
$$\phi_x(y) \leq \phi_x(z) + \nabla \phi_x(z)^T (y - z) + \frac{L}{2} ||y - z||^2$$

If f is convex, we get from the first-order characterization of convexity that ϕ_x obtains its optimum in x. The right-hand-side of the equation is a quadratic function of y whose gradient is $\nabla \phi_x(z) + L(y - z)$ so its optimum is obtained at $y^* = z - \frac{\nabla \phi_x(z)}{L}$.

Taking minimization with respect to y on both sides leads to

$$\phi_x(x) \leq \phi_x(z) - \nabla\phi_x(z)^T(\frac{\nabla\phi_x(z)}{L}) + \frac{1}{2L}||\nabla\phi_x(z)||^2$$
$$\phi_x(z) - \phi_x(x) \geq \frac{1}{L}||\nabla\phi_x(z)||^2 - \frac{1}{2L}||\nabla\phi_x(z)||^2$$
$$f(z) - f(x) - \nabla f(x)^T(z-x) \geq \frac{1}{2L}||\nabla f(z) - \nabla f(x)||^2$$

Lemma 2. *Results of one step SGD Assume* ***H.1*** *and* ***H.2****. If $\eta_t \leq \frac{1}{4L}$, we have*

$$||\bar{v}_{t+1} - w^*||^2 \leq (1-\mu\eta_t)||\bar{w}_t - w^*||^2 + 2\sum_{k=1}^{K} p_k||w_t^k - \bar{w}_t||^2 + 6L\eta_t^2\Gamma + \eta_t^2||g_t - \bar{g}_t||^2$$
$$+ 2\eta_t\sqrt{[(1-\mu\eta_t)||\bar{w}_t - w^*||^2 + 2\sum_{k=1}^{K} p_k||w_t^k - \bar{w}_t||^2 + 6L\eta_t^2\Gamma \cdot ||g_t - \bar{g}_t||^2}$$

where $\Gamma = F^* - \sum_{k=1}^{K} p_k F_k^* \geq 0$

Proof
We have $\bar{v}_{t+1} = \bar{w}_t - \eta_t g_t$. Then,

$$\begin{aligned}||\bar{v}_{t+1} - w^*||^2 &= ||\bar{w}_t - \eta_t g_t - w^* - \eta_t\bar{g}_t + \eta_t\bar{g}_t||^2 \\ &= \underbrace{||\bar{w}_t - w^* - \eta_t\bar{g}_t||^2}_{A_1} + \underbrace{2\eta_t\langle\bar{w}_t - w^* - \eta_t\bar{g}_t, \bar{g}_t - g_t\rangle}_{A_2} + \eta_t^2||g_t - \bar{g}_t||^2\end{aligned} \tag{3}$$

First, we focus on bounding $A1$. This part is exactly the same as in the work of [LHY+20].

$$A_1 = ||\bar{w}_t - w^* - \eta_t\bar{g}_t||^2 = ||\bar{w}_t - w^*||^2 \underbrace{-2\eta_t\langle\bar{w}_t - w^*, \bar{g}_t\rangle}_{B_1} + \underbrace{\eta_t^2||\bar{g}_t||^2}_{B_2} \tag{4}$$

From the L-smoothness of $F_k(\cdot)$ (**H.1** and Lemma 1), it follows that

$$\begin{aligned}||\nabla F_k(w_t^k)||^2 &\leq 2L(F_k(w_t^k) - F_k^*), \\ ||\nabla F_{S_k}(w_t^k)||^2 &\leq 2L(F_{S_k}(w_t^k) - F_{S_k}^*).\end{aligned} \tag{5}$$

By the convexity of $||\cdot||^2$ and equation (5), we have

$$B_2 = \eta_t^2||\bar{g}_t||^2 \leq \eta_t^2\sum_{k=1}^{K} p_k||\nabla F_k(w_t^k)||^2 \leq 2L\eta_t^2\sum_{k=1}^{K} p_k(F_k(w_t^k) - F_k^*).$$

Then,

$$\begin{aligned}B_1 &= -2\eta_t\langle\bar{w}_t - w^*, \bar{g}_t\rangle = -2\eta_t\sum_{k=1}^{K} p_k\langle\bar{w}_t - w^*, \nabla F_k(w_t^k)\rangle \\ &= -2\eta_t\sum_{k=1}^{K} p_k\langle\bar{w}_t - w_t^k, \nabla F_k(w_t^k)\rangle - 2\eta_t\sum_{k=1}^{K} p_k\langle w_t^k - w^*, \nabla F_k(w_t^k)\rangle\end{aligned} \tag{6}$$

For the first term,

$$\begin{aligned}-2\eta_t \sum_{k=1}^{K} p_k \langle \bar{w}_t - w_t^k, \nabla F_k(w_t^k)\rangle &\leq 2||\bar{w}_t - w_t^k||||\nabla F_k(w_t^k)|| \text{ by Cauchy-Schwarz inequality,} \\ &\leq \frac{1}{\eta_t}||\bar{w}_t - w_t^k||^2 + \eta_t||\nabla F_k(w_t^k)||^2 \text{ by AM-GM inequality.}\end{aligned} \tag{7}$$

For the second term, by the μ-strong convexity of $F_k(\cdot)$, we have

$$-\langle w_t^k - w^*, \nabla F_k(w_t^k)\rangle \leq -(F_k(w_t^k) - F_k(w^*)) - \frac{\mu}{2}||w_t^k - w^*||^2. \tag{8}$$

By combining equations (4), (6), (7) and (8) and using equation (5) again and by using the convexity of $||.||^2$ with the inequality of Jensen, it follows that

$$\begin{aligned}A_1 = ||\bar{w}_t - w^* - \eta_t \bar{g}_t||^2 \leq\ & ||\bar{w}_t - w^*||^2 + 2L\eta_t^2 \sum_{k=1}^{K} p_k(F_k(w_t^k) - F_k^*) \\ & + \eta_t \sum_{k=1}^{K} p_k(\frac{1}{\eta_t}||\bar{w}_t - w_t^k||^2 + \eta_t||\nabla F_k(w_t^k)||^2) \\ & - 2\eta_t \sum_{k=1}^{K} p_k((F_k(w_t^k) - F_k(w^*)) + \frac{\mu}{2}||w_t^k - w^*||^2) \\ & = (1 - \mu\eta_t)||\bar{w}_t - w^*||^2 + \sum_{k=1}^{K} p_k||\bar{w}_t - w_t^k||^2 \\ & \underbrace{-2\eta_t \sum_{k=1}^{K} p_k((F_k(w_t^k) - F_k^*) + 4L\eta_t^2 \sum_{k=1}^{K} p_k((F_k(w_t^k) - F_k(w^*))}_{C}\end{aligned}$$

To bound C, we define $\gamma_t = 2\eta_t(1 - 2L\eta_t)$. Given $\eta_t \leq \frac{1}{4L}$, $\eta_t \leq \gamma_t \leq 2\eta_t$. Then,

$$\begin{aligned}C &= -2\eta_t(1 - 2L\eta_t) \sum_{k=1}^{K} p_k(F_k(w_t^k) - F_k^*) + 2\eta_t \sum_{k=1}^{K} p_k(F_k(w^*) - F_k^*) \\ &= -\gamma_t \sum_{k=1}^{K} p_k(F_k(w_t^k) - F^*) + (2\eta_t - \gamma_t)(F^* - \sum_{k=1}^{K} p_k F_k^*) \\ &= -\gamma_t \sum_{k=1}^{K} p_k(F_k(w_t^k) - F^*) + 4L\eta_t^2\Gamma\end{aligned}$$

where $\Gamma = \sum_{k=1}^K p_k(F^* - F_k^*)$. Then, we use

$$\begin{aligned}\sum_{k=1}^K p_k(F_k(w_t^k) - F^*) &= \sum_{k=1}^K p_k(F_k(w_t^k) - F_k(\bar{w}_t)) + \sum_{k=1}^K p_k(F_k(\bar{w}_t - F^*) \\ &\geq \sum_{k=1}^K p_k\langle\nabla F_k(\bar{w}_t), w_t^k - \bar{w}_t\rangle + F(\bar{w}_t) - F^* \\ &\geq -\frac{1}{2}\sum_{k=1}^K p_k[\eta_t||\nabla F_k(\bar{w}_t)||^2 + \frac{1}{\eta_t}||w_t^k - \bar{w}_t||^2] + F(\bar{w}_t) - F^* \\ &\geq -\sum_{k=1}^K p_k[\eta_t L(F_k(\bar{w}_t) - F_k^*)\frac{1}{2\eta_t}||w_t^k - \bar{w}_t||^2] + F(\bar{w}_t) - F^*\end{aligned}$$

where the first inequality results from the convexity of $F_k(\cdot)$, the second from AM-GM inequality and the third from eqn. (5).

Therefore, given that $0 \leq \eta_t \leq \frac{1}{4L}$, we have $\eta_t L - 1 \leq \frac{3}{4} \leq 0$ and $0 \leq \gamma_t \leq 2\eta_t$.

$$\begin{aligned}C &\leq \gamma_t\sum_{k=1}^K p_k[\eta_t L(F_k(\bar{w}_t) - F_k^*)\frac{1}{2\eta_t}||w_t^k - \bar{w}_t||^2] - \gamma_t(F(\bar{w}_t) - F^*) + 4L\eta_t^2\Gamma \\ &\leq \underbrace{\gamma_t}_{\geq 0}\underbrace{(\eta_t L - 1)}_{\leq 0}\sum_{k=1}^K\underbrace{p_k(F_k(\bar{w}_t) - F^*)}_{\geq 0} + (4L\eta_t^2 + \gamma_t\eta_t L)\Gamma + \frac{\gamma_t}{2\eta_t}\sum_{k=1}^K p_k||w_t^k - \bar{w}_t||^2 \\ &\leq 6L\eta_t^2\Gamma + \sum_{k=1}^K p_k||w_t^k - \bar{w}_t||^2\end{aligned}$$

Recalling the expression of A_1 and plugging C into it, we have

$$\begin{aligned}A_1 &= ||\bar{w}_t - w^* - \eta_t\bar{g}_t||^2 \\ &\leq (1 - \mu\eta_t)||\bar{w}_t - w^*||^2 + 2\sum_{k=1}^K p_k||w_t^k - \bar{w}_t||^2 + 6L\eta_t^2\Gamma \qquad (9)\end{aligned}$$

In [LHY+20], $\mathbb{E}_\xi[A_2] = 0$ where ξ refers to the random batch. This is not the case here as we are not using all the data in a single epoch.

$$\begin{aligned}A_2 &= 2\eta_t\langle\bar{w}_t - w^* - \eta_t\bar{g}_t, \bar{g}_t - g_t\rangle \\ &\leq 2\eta_t||\bar{w}_t - w^* - \eta_t\bar{g}_t|| \cdot ||\bar{g}_t - g_t|| \text{ using Cauchy-Schwarz inequality} \\ &\leq 2\eta_t\sqrt{A_1}\sqrt{||\bar{g}_t - g_t||^2} \qquad (10)\end{aligned}$$

Putting equations (3), (9), (10) together, we have:

$$\begin{aligned}||\bar{v}_{t+1} - w^*||^2 &\leq A_1 + \eta_t^2||g_t - \bar{g}_t||^2 + 2\eta_t\sqrt{A_1}\sqrt{||\bar{g}_t - g_t||^2} \\ &\leq (1 - \mu\eta_t)||\bar{w}_t - w^*||^2 + 2\sum_{k=1}^K p_k||w_t^k - \bar{w}_t||^2 + 6L\eta_t^2\Gamma + \eta_t^2||g_t - \bar{g}_t||^2 \\ &+ 2\eta_t\sqrt{[(1 - \mu\eta_t)||\bar{w}_t - w^*||^2 + 2\sum_{k=1}^K p_k||w_t^k - \bar{w}_t||^2 + 6L\eta_t^2\Gamma] \cdot ||g_t - \bar{g}_t||^2}\end{aligned}$$

Lemma 3. *Bounding the variance Assume* **H.3***. Then,*

$$||g_t - \bar{g}_t||^2 \leq \sum_{k=1}^{K} p_k^2 \sigma_k^2$$

$$\begin{aligned}
||\bar{g}_t - g_t||^2 &= ||\sum_{i=1}^{K} p_k(\nabla F_k(w_t^k) - \nabla F_{S_k}(w_t^k))||^2 \\
&\leq \sum_{i=1}^{K} ||p_k(\nabla F_k(w_t^k) - \nabla F_{S_k}(w_t^k))||^2 \\
&\leq \sum_{k=1}^{K} p_k^2 \sigma_k^2 \text{ using } \mathbf{H.3}.
\end{aligned}$$

Lemma 4. *Bounding the divergence of $\{w_t^k\}$. Assume* **H.4***, that η_t is non-increasing and $\eta_t \leq 2\eta_{t+E}$ for all $t \geq 0$. It follows that*

$$\sum_{i=1}^{K} p_k ||\bar{w}_t - w_t^k||^2 \leq 4\eta_t^2 (E-1)^2 G^2$$

Proof. The proof is the same as the one by [LHY⁺20]. For any $t \geq 0$, there exists a $t_0 \leq t$, such that $t - t_0 \leq E - 1$ and $w_{t_0}^k = \bar{w}_{t_0}$ for all $k = 1, 2, \ldots, N$. Also, we use the fact that η_t is non-increasing and $\eta_{t_0} \leq 2\eta_t$ for all $t - t_0 \leq E - 1$ then

$$\begin{aligned}
\sum_{k=1}^{N} p_k ||\bar{w}_t - w_t^k||^2 &= \sum_{k=1}^{N} p_k ||(w_t^k - \bar{w}_{t_0}) - (\bar{w}_t - \bar{w}_{t_0})||^2 \\
&\leq \sum_{k=1}^{N} p_k ||w_t^k - \bar{w}_{t_0}||^2 \\
&\leq \sum_{k=1}^{N} p_k (E-1) \sum_{t=t_0}^{t-1} \eta_t^2 ||F_{S_k}(w_t^k)||^2 \\
&\leq \sum_{k=1}^{N} p_k (E-1) \sum_{t=t_0}^{t-1} \eta_{t_0}^2 G^2 \\
&\leq \sum_{k=1}^{N} p_k (E-1)^2 \eta_{t_0}^2 G^2 \\
&\leq (E-1)^2 \eta_{t_0}^2 G^2 \\
&\leq 4\eta_t^2 (E-1)^2 G^2
\end{aligned}$$

For the first inequality, we use $\mathbb{E}(||X - \mathbb{E}(X)||^2) \leq \mathbb{E}(||X||^2)$ where $X = w_t^k - \bar{w}_{t_0}$ with probability p_k. In the second inequality, we use Jensen inequality:

$$||w_t^k - \bar{w}_{t_0}||^2 = ||\sum_{t=t_0}^{t-1} \eta_t \nabla F_{S_k}(w_t^k)||^2 \leq (t - t_0) \sum_{t=t_0}^{t-1} \eta_t^2 ||F_{S_k}(w_t^k)||^2$$

In the third inequality, we use $\eta_t \leq \eta_{t_0}$ and **H.4**. In the last inequality, we use $\eta_{t_0} \leq 2\eta_t$ for all $t - t_0 \leq E - 1$.

Theorem 2. *Assume H.1 to H.4. Choose $\gamma \geq max\{\frac{8L}{\mu}, E\}$ and $\eta_t = \frac{2}{\mu(t+\gamma)}$ for all $t \geq 0$. Then,*

$$F(\bar{w}_t) - F^* \leq \frac{L}{2}\frac{v}{t+\gamma}.$$

where $v = max\{v_+, (\gamma+1)\Delta_1\}$ and
$v_+ = \frac{\beta^2 B + 2\beta^2\sqrt{CD}}{\mu\beta - 1} + \frac{(2\beta\sqrt{D})^2 + \sqrt{(2\beta\sqrt{D})^4 + 4(\beta^2 B + 2\beta^2\sqrt{CD})(\mu\beta-1)(2\beta\sqrt{D})^2}}{2(\mu\beta-1)^2}$.

Proof. We always have $\bar{w}_{t+1} = \bar{v}_{t+1}$. Let $\Delta_t = ||\bar{w}_t - w^*||^2$. From Lemma 2, Lemma 3 and Lemma 4, it follows that

$$\Delta_{t+1} \leq (1-\mu\eta_t)\Delta_t + \eta_t^2 B + 2\eta_t\sqrt{[(1-\mu\eta_t)\Delta_t + \eta_t^2 C]\cdot D} \tag{11}$$

where $C = 8(E-1)^2G^2 + 6L\Gamma$, $D = \sum_{k=1}^{K} p_k^2\sigma_k^2$ and $B = C + D$.

Assume $\eta_t = \frac{\beta}{t+\gamma}$ with $\beta \geq \frac{1}{\mu}$. We want to prove that for all t, $\Delta_t \leq \frac{v}{t+\gamma}$.

It holds for $t = 1$ if we choose $v \geq (\gamma+1)\Delta_1$. Then we prove it by induction by assuming that $\Delta_t \leq \frac{v}{t+\gamma}$ for some t.

$$\begin{aligned}
\Delta_{t+1} &\leq (1-\mu\eta_t)\Delta_t + \eta_t^2 B + 2\eta_t\sqrt{[(1-\mu\eta_t)\Delta_t + \eta_t^2 C]\cdot D} \\
&\leq (1-\mu\frac{\beta}{t+\gamma})\frac{v}{t+\gamma} + (\frac{\beta}{t+\gamma})^2 B + 2\frac{\beta}{t+\gamma}\sqrt{[(1-\mu\frac{\beta}{t+\gamma})\frac{v}{t+\gamma} + (\frac{\beta}{t+\gamma})^2 C]\cdot D} \\
&\leq \frac{v}{\gamma+t} + \frac{1}{(\gamma+t)^2}(\beta^2 B - \mu\beta v) + 2\frac{\beta}{(t+\gamma)^2}\sqrt{[(t+\gamma-\mu\beta)v + \beta^2 C]\cdot D} \\
&\leq \frac{v}{\gamma+t} + \frac{1}{(\gamma+t)^2}(\beta^2 B - \mu\beta v) + 2\frac{\beta}{(t+\gamma)^2}\sqrt{[(t+\gamma-1)v + \beta^2 C]\cdot D} \\
&\leq \frac{v}{\gamma+t} + \frac{1}{(\gamma+t)^2}(\beta^2 B + 2\beta^2\sqrt{CD} - \mu\beta v) + 2\beta\sqrt{D}\frac{\sqrt{(t+\gamma-1)v}}{(t+\gamma)^2} \\
&\leq \frac{v}{\gamma+t} + \frac{1}{(\gamma+t)^2}(\beta^2 B + 2\beta^2\sqrt{CD} - \mu\beta v) + 2\beta\sqrt{D}\frac{\sqrt{v}}{(t+\gamma)^{3/2}} \\
&\leq \frac{v}{\gamma+t} - \frac{v}{(\gamma+t)^2} + \frac{1}{(\gamma+t)^{3/2}}(\beta^2 B + 2\beta^2\sqrt{CD} - (\mu\beta-1)v + 2\beta\sqrt{D}\sqrt{v})
\end{aligned}$$

For the first part, we have

$$\begin{aligned}
\frac{v}{\gamma+t} - \frac{v}{(\gamma+t)^2} &= \frac{(\gamma+t-1)v}{(\gamma+t)^2} \\
&\leq \frac{(\gamma+t-1)v}{(\gamma+t)^2-1} \\
&= \frac{v}{\gamma+t+1}
\end{aligned}$$

For the second part, we would like $\beta^2 B + 2\beta^2\sqrt{CD} - (\mu\beta-1)v + 2\beta\sqrt{D}\sqrt{v} \leq 0$ as this would imply that $\Delta_{t+1} \leq \frac{v}{t+1+\gamma}$. We have an equation in the form $a - b\cdot v \leq -c\sqrt{v} \leq 0$ where a, b and c are strictly positive. This implies

$$\begin{aligned}
a^2 + b^2\cdot v^2 - 2ab\cdot v &\geq c^2\cdot v \\
b^2\cdot v^2 - (2ab + c^2)\cdot v + a^2 &\geq 0
\end{aligned}$$

$$\begin{aligned}\Delta &= (2ab + c^2)^2 - 4a^2b^2 \\ &= 4a^2b^2 + c^4 + 4abc^2 - 4a^2b^2 \\ &= c^4 + 4abc^2 \geq 0\end{aligned}$$

$$\begin{aligned}v_{+/-} &= \frac{2ab + c^2 \pm \sqrt{\Delta}}{2b^2} \\ &= \frac{2ab + c^2 \pm \sqrt{c^4 + 4abc^2}}{2b^2}\end{aligned}$$

$$\begin{aligned}a - b \cdot v_+ &= a - \frac{2ab + c^2 + \sqrt{c^4 + 4abc^2}}{2b} \\ &= -\frac{c^2 + \sqrt{c^4 + 4abc^2}}{2b} \leq 0\end{aligned}$$

So if we choose $v \geq v_+ \geq 0$, we have $(a - b \cdot v)^2 \geq (-c\sqrt{v})^2$ and $a - b \cdot v \leq 0$ and $-c\sqrt{v} \leq 0$. This leads to $a - b \cdot v \leq -c\sqrt{v} \leq 0$.

In our case, $a = \beta^2 B + 2\beta^2\sqrt{CD}$, $b = \mu\beta - 1$, $c = 2\beta\sqrt{D}$. So,

$$\begin{aligned}v_+ &= \frac{2ab + c^2 + \sqrt{c^4 + 4abc^2}}{2b^2} \\ &= \frac{\beta^2 B + 2\beta^2\sqrt{CD}}{\mu\beta - 1} + \frac{(2\beta\sqrt{D})^2 + \sqrt{(2\beta\sqrt{D})^4 + 4(\beta^2 B + 2\beta^2\sqrt{CD})(\mu\beta - 1)(2\beta\sqrt{D})^2}}{2(\mu\beta - 1)^2}\end{aligned}$$

If we choose $v = \max\{v_+, (\gamma + 1)\Delta_1\}$, we have $\Delta_t \leq \frac{v}{t+\gamma}$.

Then, by the L-smoothness of $F(\cdot)$, we have

$$F(\bar{w}_t) - F^* \leq \frac{L}{2}||\bar{w}_t - w^*||^2 = \frac{L}{2}\Delta_t \leq \frac{L}{2}\frac{v}{t + \gamma}.$$

B Summary of experimental results

Configuration	Results	Conclusion	Figures
Centralized	Random outperforms CRAIG for selecting 25000 out of 50000 imgs	Submodular optimization is not interesting for big subsets	Fig. 3.1
Centralized	CRAIG slightly outperforms random for smaller subsets	Submodular optimization can outperform random sampling for smaller subsets	Fig. 3.2 &. 3.3
Centralized	Combining random and CRAIG does not hurt the accuracy and reduces the runtime	Random sampling and submodular optimization can be combined to speed up the latter	Fig. 3.3 & 3.4
Centralized	Using per-class selection does not hurt the accuracy	Per-class selection can be implemented to reduce runtime	Fig. 3.5
Federated, 1 machine, IID, CPU	Using the full set is better in terms of rounds	Using subsets is not interesting for IID data	Section 4.2.1
Federated, 1 machine, CPU	The training on full set is very slow in terms of runtime	Using subsets can speed up training but the results are skewed due to parallelization problems	Section 4.2.1 & 4.2.2
Federated, 1 machine per client, CPU	The training on full set is very slow in terms of runtime	Using subsets can speed up training for clients with limited system resources	Fig. 4.12
Federated, GPU	Using subsets outperforms using the full set	Too many local updates can hurt performance, subsets are relevant in that case	Fig 4.13
Federated, GPU	The runtime for CRAIG is too long compared with random or full set	Submodular maximization is not relevant when the subset selection step takes much more time than the training in itself	Fig. 4.13 & 4.14
Federated, GPU	Reducing the frequency we make the data summaries does not necessarily hurt performance	Reducing that frequency can help reduce runtime for submodular maximization	Fig. 4.15 & 4.16
Federated, GPU	An optimal size for the data summary can be found	A balance has to be found between accuracy and runtime	Fig. 4.17

Table B.1: Recap of experimental results.

Bibliography

[AASC19] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers, 2019.

[ai.18] ai.google. Under the hood of the pixel 2: How ai is supercharging hardware. *Available at https://archive.ph/o4dS4*, 2018. Last accessed 11 September 2023.

[AZK⁺19] Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *ArXiv*, abs/1906.03671, 2019.

[BTM⁺20] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[CB19] Luca Corinzia and Joachim M. Buhmann. Variational federated multi-task learning. *CoRR*, abs/1906.06268, 2019.

[CCA⁺21] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[CCPV11] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[CKMT18] Sebastian Caldas, Jakub Konečný, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *CoRR*, abs/1812.07210, 2018.

[CNSR20] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, Los Alamitos, CA, USA, dec 2020. IEEE Computer Society.

[CSZZ20] L. Cui, X. Su, Y. Zhou, and L. Zhang. Clustergrad: Adaptive gradient compression by clustering in federated learning. In *2020 IEEE Global Communications Conference, 2020, pp. 1-7*, 2020.

[CVZ14] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

[CYM+20] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations*, 2020.

[DMPHO21] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Compute and energy consumption trends in deep learning inference. *CoRR*, abs/2109.05472, 2021.

[DTN20] Canh T. Dinh, Nguyen H. Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[EKM+19] Hubert Eichner, Tomer Koren, Brendan Mcmahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1764–1773. PMLR, 09–15 Jun 2019.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, jul 1998.

[flo23] Flower github repository. *Available at* `https://github.com/adap/flower`, 2023. Last accessed 30 August 2023.

[GCYR20] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[GHYR19] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment, 2019.

[GLS81] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, pages 169–197, 1981.

[GZB22] Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. In *Database and Expert Systems Applications: 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part I*, page 181–195, Berlin, Heidelberg, 2022. Springer-Verlag.

[Hao19] Karen Hao. How apple personalizes siri without hoovering up your data. *Available at* `https://www.technologyreview.com/2019/12/11/131629/apple-ai-personalizes-siri-federated-learning/`, 2019. Last accessed 11 September 2023.

[HHR+22] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning, 2022.

[HKMM21] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. *PMLR*, 130:2350–2358, 2021.

[HRM$^+$19] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[JA18] P. Jiang and G. Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In *NeurIPS*, 2018.

[JKRK19] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic metalearning. *CoRR*, abs/1909.12488, 2019.

[JRSB19] K. J. Joseph, Vamshi Teja R., Krishnakant Singh, and Vineeth N. Balasubramanian. Submodular batch selection for training deep neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 2677–2683. AAAI Press, 2019.

[KBT19] Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. *Adaptive Gradient-Based Meta-Learning Methods.* Curran Associates Inc., Red Hook, NY, USA, 2019.

[KG05] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 324–331, Arlington, Virginia, USA, 2005. AUAI Press.

[KIK$^+$19] V. Kaushal, R. Iyer, S. Kothawade, R. Mahadev, K. Doctor, and G. Ramakrishnan. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1289–1299, Los Alamitos, CA, USA, jan 2019. IEEE Computer Society.

[KKM$^+$20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 13–18 Jul 2020.

[KMA$^+$21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh

Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2021.

[KMR19] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Better communication complexity for local SGD. *CoRR*, abs/1909.04746, 2019.

[KMY$^+$16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[KRI22] Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh Iyer. Submodlib: A submodular optimization library, 2022.

[KSR$^+$20] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, Rishabh Iyer University of Texas at Dallas, Indian Institute of Technology Bombay Institution One, and IN Two. Glister: Generalization based data subset selection for efficient and robust learning. In *AAAI Conference on Artificial Intelligence*, 2020.

[KSR$^+$21] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, Abir De, and Rishabh K. Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, 2021.

[KST13] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Mathematics of Operations Research*, 38(4):729–739, 2013.

[KWHCJ19] Bong Jun Ko, Shiqiang Wang, Ting He, and Dave Conway-Jones. On data summarization for machine learning in multi-organization federations. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 63–68, 2019.

[LHD$^+$20] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Differentially private asynchronous federated learning for mobile edge computing in urban informatics. *IEEE Transactions on Industrial Informatics*, 16(3):2134–2143, 2020.

[LHM$^+$18] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

[LHY$^+$20] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data, 2020.

[LKCT19] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. *CoRR*, abs/1909.05830, 2019.

[LSZ$^+$18] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2018.

[LZCW21] Kai Liang, Huiru Zhong, Haoning Chen, and Youlong Wu. Wyner-ziv gradient compression for federated learning. *CoRR*, abs/2111.08277, 2021.

[MBK+15] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 1812–1818. AAAI Press, 2015.

[MBL20] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[Mir17] B. Mirzasoleiman. *Big Data Summarization Using Submodular Functions*. PhD thesis, ETH Zurich, 2017.

[MMR+16] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. 2016.

[Moi16] Ankur Moitra. Lecture notes in advanced algorithms. *http://people.csail.mit.edu/moitra/docs/6854lec13.pdf*, March 2016. Last accessed 19 April 2023.

[MVBA21] Katerina Margatina, Giorgos Vernikos, Loïc Barrault, and Nikolaos Aletras. Active learning by acquiring contrastive examples, 2021.

[MW21] Luke Melas-Kyriazi and Franklyn Wang. Intrinisic gradient compression for federated learning. *CoRR*, abs/2112.02656, 2021.

[NWF78] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - i. *Mathematical Programming*, 1978.

[PGL+21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

[PSM+21] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, Sudeep Agarwal, Julien Freudiger, Andrew Byde, Abhishek Bhowmick, Gaurav Kapoor, Si Beaumont, Áine Cahill, Dominic Hughes, Omid Javidbakht, Fei Dong, Rehan Rishi, and Stanley Hung. Federated evaluation and tuning for on-device personalization: System design & applications, 2021.

[QPB+21] Xinchi Qiu, Titouan Parcollet, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. Can federated learning save the planet?, 2021.

[QPFM+23] Xinchi Qiu, Titouan Parcollet, Javier Fernandez-Marques, Pedro Porto Buarque de Gusmao, Yan Gao, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. A first look into the carbon footprint of federated learning, 2023.

[RCZ+21] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[SBN20] Jacob Schreiber, Jeffrey Bilmes, and William Stafford Noble. apricot: Submodular selection for data summarization in python. *Journal of Machine Learning Research*, 21(161):1–6, 2020.

[SCST18] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. *CoRR*, abs/1705.10467, 2018.

[SDSE19] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai, 2019.

[SF11] Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011.

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[Sin16] Yaron Singer. Lecture notes in advanced optimization. *Available at https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture24.pdf*, April 2016. Last accessed 20 April 2023.

[SMS21] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3710–3722, 2021.

[SS18] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018.

[sup23] support.google. How messages improves suggestions with federated technology. *Available at https://support.google.com/messages/answer/9327902*, 2023. Last accessed 11 September 2023.

[SWMS20] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413, 2020.

[Wei22] Matt Weinberg. Lecture notes in advanced algorithm design - lecture 7: Submodular functions, lovász extension and minimization. *https://www.cs.princeton.edu/ hy2/teaching/fall22-cos521/notes/SFM.pdf*, September 2022. Last accessed 19 April 2023.

[WIB14] Kai Wei, Rishabh K. Iyer, and Jeff A. Bilmes. Fast multi-stage submodular maximization. In *International Conference on Machine Learning*, 2014.

[WIB15] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1954–1963, Lille, France, 07–09 Jul 2015. PMLR.

[WJC$^+$19] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu 0011, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80savings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Edward A. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5139–5151, 2019.

[WLL$^+$20] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[WWL$^+$21] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature Communications*, 13(1), apr 2021.

[WYS$^+$20] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations 2020*, ICLR'20, 2020.

[XKG20] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. In *12th Annual Workshop on Optimization for Machine Learning*, OPT'20, 2020.

[YAG$^+$19] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, Cambridge, MA, USA, 2019. Curran Associates Inc.

[YL22] Xiao-Tong Yuan and Ping Li. On convergence of fedprox: Local dissimilarity invariant bounds, non-smoothness and beyond, 2022.

[YYZ18] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning, 2018.

[ZLL$^+$18] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018.