Data Summarization for Federated Learning

Julianna Devillers, Olivier Brun, Balakrishna Prabhu

Data Summarization for Federated Learning *

Julianna Devillers[1,2], Olivier Brun[2], and Balakrishna J. Prabhu[2]

[1] ISAE-SUPAERO, Toulouse, France
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
Julianna.DEVILLERS@student.isae-supaero.fr
{brun,Balakrishna.Prabhu}@laas.fr

Abstract. We explore data summarization techniques as a mean to reduce the energy footprint of Federated Learning (FL). We formulate the problem of selecting a small subset of data points that best represent the gradient of each local dataset as a submodular maximization problem and provide sufficient conditions under which the FL training is guaranteed to converge to the same global model as if the whole local datasets have been used on each client. Experimental results on IID and non-IID datasets show that this approach yields a similar accuracy as training on the full local datasets, but with a significant reduction of runtimes. There is however no clear advantage of data summarization over random sampling.

Keywords: Data summarization, FedAvg, convergence

1 Introduction

Training state-of-the-art deep learning models has an ever increasing computational cost, which double every few months, and consumes a lot of energy [20,21]. For instance, it is known that training the GPT-3 language model (175 billion parameters) consumed around 1,287 MWh of energy, which represents an amount of energy equivalent to the yearly consumption of 126 Danish homes and a CO_2 emission equivalent to 552 round-trip flights between New York and Paris [17]. It is now broadly recognized that the current race for improved model accuracy in ML is not sustainable both from an environmental and economic point of view.

There is an increasing interest in a new distributed ML paradigm called Federated Learning (FL) in which many clients collaboratively train a shared model under the orchestration of a central server, while keeping the training data private. At each round, the central server selects a subset of clients and sends them the parameters of the shared model. The selected clients train the model on their local data and send back their gradient information to the central server for aggregation and update of the shared model. FL has a wide range of applications,

* This work was partially financed by French Agence Nationale de la Recherche (ANR) through grant ANR-22-CE23-0024 (project DELIGHT).

especially for sensitive data for example in the fields of healthcare or finance or considering data on peoples' localization. FL has however a surprisingly large carbon footprint, which can even be up to two orders of magnitude higher than its more traditional centralized counterparts [18]. As FL is becoming more and more prevalent, the reduction of its energy footprint therefore becomes a real issue.

A first approach aims at reducing the energy spent for the communications between the clients and the central server. Gradient sparsification and gradient quantization techniques (see, e.g., [7,3,6]) have been recently proposed for that purpose. Since compression is a lossy process, the gains in terms of communication costs are usually achieved at the expense of a worse iteration complexity and there is not a good understanding of how these techniques impact the total energy consumption and whether they are worth applying in general.

Another approach, which is explored in this paper, aims at reducing the energy spent for local model updates by using data summarization techniques. The general idea is to extract a small representative set from the original local dataset on each client and to train on that smaller set. 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. This raises however two main questions. The first one is: "*how to choose the data summary on each client while preserving data privacy and converging to the same global model as if the whole local datasets have been used?*". The second one is: "*is he extra energy required for data summarization offset by the energy saved during the training phase?*". These two questions are investigated in this paper.

1.1 Contributions

From a theoretical point of view, we propose to constitute the data summary on each client by choosing the data points in the local dataset which best represent the local gradient of the loss function. We show that the computation of the data summary can be cast as a submodular maximization problem or as a submodular cover problem. With the latter setup for data summarization, we provide sufficient conditions under which the FL training is guaranteed to converge to the same global model as if the whole local datasets have been used on each client.

From a practical point of view, we study the empirical test accuracy and the efficiency of the gradient matching method based on submodular maximization. Unfortunately, it is difficult to estimate the energy cost of our experiments and we use the number of rounds before convergence and the runtime instead. We note that runtime is a good proxy for the computational cost, although it does not capture the costs of communication in a real federated environment. We use the Flower toolkit to train Deep Neural Networks (DNN) on the MNIST and CIFAR10 datasets and evaluate these performance metrics in scenarios with IID and non-IID datasets. They are compared against that of the FL with the full datasets, and against random sampling. In contrast to [15], our experimen-

tal results did not show a clear advantage of data summarization over random sampling.

1.2 Organization

In Sec. 3, a federated learning algorithm that uses data summary is first presented, and its convergence guarantee is proven. Results from numerical experiments are detailed in Sec. 4. A discussion on our observations from these experiements as well as future work is given in Sec. 5.

Before presenting the algorithm in Sec. 3, we start by presenting in the next section the related work as well as the concepts from federated learning and submodular maximization that will be useful later on.

2 Preliminaries

In this section, we will summarize the two main objects of this paper. FL algorithms and data summarization techniques. First, we will present various FL algorithms and their convergence rates. This will be followed by known results in data summarization. These concepts will be used in the next section to show the convergence rate of a specific FL algorithm when it is combined with data summarization.

Federated learning (FL) [13] 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. It is a distributed learning paradigm with two key challenges that differentiate it from traditional distributed optimization: (1) significant variability in terms of the characteristics (hardware, data rate, etc.) of each device or client in the network (system heterogeneity), and (2) non-identically distributed data across the devices (statistical heterogeneity) which can lead to model bias during training.

In the FL 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 $n_k = |\mathcal{P}_k|$. The objective of the server is the following optimization problem:

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

where $f_i(w) = \ell(x_i; y_i, w)$ is the loss of the prediction on example $(x_i; y_i)$ made with the model parameters w.

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. The IID assumption is not always satisfied in FL settings since the server has no control on how the data is distributed among the clients. The degree of heterogeneity

of the data in this non-IID setting can be quantified by $\Gamma = F^* - \sum_k p_k F_k^*$, where F^* (resp. F_k^*) is the optimal loss on the whole (resp. local) dataset and $p_k = n_k/n$ [12].

One of the first algorithm specifically designed for FL was FedAvg [13] which proposed to cooperatively train a global model. The algorithm works in discrete-time steps or rounds. At the beginning of each round, the clients receive the parameters of the global model from the central server. They then update the parameters using some standard learning algorithm on their local dataset and send the updated parameters back to the central server at the end of the round. The server averages the received model parameters and sends them to the end devices to signal the start of a new round. The pseudo-code of FedAvg is given in Alg. 1 with w_t being the parameters of current model at round t. 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 mini-batch size used for the client updates.

Algorithm 1 FedAvg algorithm

procedure SERVER(C, E, B) ▷ Run on central server
  Initialize w_0
  for each round $t = 1, 2, \dots, T$ **do**
    $m \leftarrow \max(C\,K, 1)$
    $S_t \leftarrow$ random set of m clients
    $N_t \leftarrow \sum_{k \in S_t} n_k$
    for each client $k \in S_t$ in parallel **do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    end for
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{N_t} w_{t+1}^k$
  end for
  return w_{T+1}
end procedure

procedure CLIENTUPDATE(k, w) ▷ Run on client k
  $\mathcal{B} \leftarrow$ split $\mathcal{P}_k$ into batches of size B
  for each local epoch $i = 1, 2, \dots, E$ **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
    end for
  end for
  return w to server
end procedure

The convergence of FedAvg has been analysed in several works [12][23][8] and the algorithm has established itself as the algorithm of choice for FL due to its simplicity and relatively low communication cost.

When there is a large statistical heterogeneity in the client datasets, the local updates in FedAvg can push the local model parameters towards the local optimum which is different from the global optimum. The global parameter models obtained from averaging can thus be different from the actual global optimum. FedProx [11] proposes to add a regularization term to the objective function at every client. The new objective is

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

with the second term on the RHS being the proximal term.

2.1 Convergence in FL

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 [12] propose a convergence analysis for FedAvg where only a subset of clients participate in the training at each round. The authors of [24] note that the assumption taken to bound gradient dissimilarity in the work introducing FedProx [11] 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 3 are some examples of convergence results along with the assumptions taken for FedAvg and FedProx. The explanations of the assumptions are in Table 1 and Table 2. Recall that here T the total number of communication rounds, E the number of local rounds between each communication round, K the total number of clients, C the number of clients participating in each round.

Table 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 relevant in certain cases.

Table 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 3. Convergence rates.

Method	Non-IID	Other assumptions	Variant	Rate
Yu et al. [23]	$(G, 0)$-BGD	SMO; BLGV; BLGN	AC	$\mathcal{O}(\frac{1}{\sqrt{KT}})$
Khaled et al. [9]	(G-B)-BGD	SMO; CVX; BLGV	AC; LBG	$\mathcal{O}(\frac{K}{T}) + \mathcal{O}(\frac{1}{\sqrt{KT}})$
Li et al. [12]	$(G, 0)$-BGD	SMO; SCVX; BLGV; BLGN	-	$\mathcal{O}(\frac{E}{T})$
Karimireddy et al. [8]	(G, B)-BGD	SMO; BLGV	-	$\mathcal{O}(\frac{T(1-C/K)}{TC})$
FedProx [11]	$(0, B)$-BGD	SMO; BNCVX	Prox	$\mathcal{O}(\frac{1}{\sqrt{T}})$

2.2 Data Summarization

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*,

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

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 [14] for other examples).

We can distinguish two different data summarization (DS) 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,

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

These optimization problems are NP-hard for many classes of submodular functions [10][5][22]. However, a simple greedy algorithm proposed by Nemhauser in [16] is known to be very effective (see Algorithm 2).

Algorithm 2 Greedy algorithm

```
1: procedure GREEDY
2:     S ← ∅
3:     while |S| < k do
4:         v ← argmax_{e∈V} Δ_f(e|S)
5:         S ← S ∪ {v}
6:     end while
7:     return S
8: end procedure
```

Theorem 1 ([16]). *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, [14] proposes an accelerated version of the greedy algorithm called Stochastic-Greedy that scales to voluminous datasets.

3 Federated learning with data summary

To reduce computational and energy costs, we propose that each client extracts a data summary S_k from its local dataset $\mathcal{P}_k$. Further, we also modify how updates are done by clients in each epoch. Instead of training on mini-batches and covering the whole local dataset, we propose that client k perform one full-batch update per epoch on S_k.

The pseudo-code of the proposed FedAvg algorithm with data summary, which we call FedAVgDS, is shown in Alg. 3. We point out two main differences with the FedAvg algorithm both of which are in the ClientUpdate procedure. First, a data summary is performed in each local epoch. And, second, the local training is performed on the whole set S_k at once unlike the batch-based training in FedAvg. Since the size of S_k is expected to be small, we think that dividing it into batches is not necessary and the gradient can be computed on S_k in it entirety.

Algorithm 3 FedAvgDS algorithm

procedure SERVER(C, E) ▷ Run on central server
  Initialize w_0
  for each round $t = 1, 2, \ldots, T$ **do**
    $m \leftarrow \max(C\,K, 1)$
    $S_t \leftarrow$ random set of m clients
    $N_t \leftarrow \sum_{k \in S_t} n_k$
    for each client $k \in S_t$ in parallel **do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    end for
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{N_t} w_{t+1}^k$
  end for
  return w_{T+1}
end procedure

procedure CLIENTUPDATE(k, w) ▷ Run on client k
  for each local epoch $i = 1, 2, \ldots, E$ **do**
    $S_k \leftarrow \text{GreedyDataSummary}(\mathcal{P}_k, \epsilon_k)$
    $w \leftarrow w - \eta \nabla F_{S_k}(w)$
  end for
  return w to server
end procedure

3.1 Convergence of FedAvgDS

We will prove convergence of FedAvgDS in the restricted setting of full node participation by mimicking the one of [12]. In fact, to show convergence of FedAvg, [12] modify slightly the original algorithm and assume that only one update is performed per epoch on a mini-batch. That is, the ClientUpdate procedure for

the modified FedAvg in [12] is the same as that of FedAvgDS except that S_k is chosen randomly and not by the GreedyDataSummary procedure.

For the convergence guarantees, we will need the same four assumptions as in [12] except that the third and the fourth ones are modified to account for the DS step as we explain below.

A.1 F_k and the F_{S_k} are all L-smooth.
A.2 F_k and the F_{S_k} are μ-strongly convex.
A.3 $\|\nabla F_k(w) - \nabla F_{S_k}(w)\| \leq \sigma_k$, $\forall w$. Since we do not draw samples randomly, we do not need expectations in our bounds.
A.4 $\|\nabla F_{S_k}(w)\| \leq G$, $\forall w$. In FedAvg, this inequality is on the second moment of ∇F computed on the random mini-batches. Again, since we train over all of S_k, we need the bound on the norm of ∇F_{S_K}.
A.5 All nodes participate in all the iterations. For a random variable X taking values x_k, $k = 1, \ldots K$, we have $\mathbb{E}[X] = \sum_{k=1}^{K} p_k x_k$ with $\sum_{k=1}^{K} p_k = 1$. For example, $p_k = \frac{n_k}{n}$.

Theorem 2. *Assume **A.1** to **A.5**. 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 $v = max\{v_+, (\gamma+1)\Delta_1\}$,

$$v_+ = \frac{\beta^2 B + 2\beta^2\sqrt{CD}}{\mu\beta - 1} + \frac{(2\beta\sqrt{D})^2 + \sqrt{(2\beta\sqrt{D})^4 + 4(\beta^2 B + 2\beta^2\sqrt{CD})(\mu\beta-1)(2\beta\sqrt{D})^2}}{2(\mu\beta-1)^2},$$

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

The proof of the above theorem is in Appendix A of [4].

3.2 Finding a good DS subset

We consider two methods for extracting S_k at client k. Both are based on distances computed from the average gradient of the loss function, and motivated from the observation that, after the local training, in an ideal setting, we would like the weights of client k to be updated as:

$$w \leftarrow w - \eta\nabla_w F_k(w), \tag{3}$$

that is, by following the average gradient of the loss on the whole local dataset $\mathcal{P}_k$. Since we want the updates with DS to closely follow those without DS, one natural way is to compute S_k such that the average gradient of the loss on S_k is close to that of $\mathcal{P}_k$. Define

$$\nabla_w F_S(w) = \frac{1}{|S|}\sum_{i\in S}\nabla_w f_i(w), \tag{4}$$

to be the average gradient on S. If on S_k, we have that $\nabla_w F_{S_k}(w)$ is a good approximation of $\nabla_w F_k(w)$, then it will lead to a fairly similar updating of weights to that of FedAvg, and we can potentially hope to obtain convergence guarantees fairly close to those of FedAvg (i.e., without DS).

Finding such a S_k corresponds to minimizing

$$\|\nabla F_k(w) - \nabla_w F_{S_k}(w)\|, \tag{5}$$

under a cardinality constraint on S_k. However, we are faced with the problem that (5) is not submodular, a property that is desirable if we want to call upon efficient DS algorithm we saw in Sec. 2.2.

A first method around this problem is to instead use the objective function as in Exemplar Based Clustering which can then be transformed into a monotone submodular function (see [14] for this method). Towards this end, define

$$L_k(S_k) = \frac{1}{|P_k|} \sum_{i \in P_k} \min_{j \in S_k} \|\nabla_w f_i(w) - \nabla_w f_j(w)\|, \tag{6}$$

which computes how far the points in set $\mathcal{P}_k$ are from their closest counterparts in S_k. Here, the distance between two points is defined through the norm of the gradients.

The quantity in (6) can be related to a slight modification of that in (5) as follows. 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 have:

$$\begin{aligned} L_k(S_k) &= \frac{1}{|P_k|} \sum_{i \in P_k} \left\|\nabla_w f_i(w) - \nabla_w f_{\sigma(i)}(w)\right\| \\ &\geq \frac{1}{|P_k|} \left\| \sum_{i \in P_k} (\nabla_w f_i(w) - \nabla_w f_{\sigma(i)}(w)) \right\| \\ &\geq \frac{1}{|P_k|} \left\| \sum_{i \in P_k} \nabla_w f_i(w) - \sum_{j \in S_k} \alpha_j \nabla_w f_j(w) \right\|. \end{aligned} \tag{7}$$

Thus, by modifying the definition of $\nabla_w F_{S_k}$ to

$$\nabla_w F_{S_k}(w) = \frac{1}{|P_k|} \sum_{i \in S_j} \alpha_j \nabla_w f_j(w), \tag{8}$$

it can inferred that if we find a S_k with $L_k(S_k) \leq \epsilon$, its norm in (5) is also at most ϵ.

To transform L_k into a monotone submodular function, we follow the steps in [14]. Define

$$g_k(S) = L_k(\{e_0\}) - L_k(S \cup \{e_0\}),$$

with e_0 a phantom point chosen such that $\max_{i' \in P_k} \|\nabla_w f_i(w) - \nabla_w f_{i'}(w)\| \leq \|\nabla_w f_i(w) - \nabla_w f_{e_0}(w)\|$, $\forall i \in P_k$. That is, e_0 is a point that is farther than any point in $\mathcal{P}_k$ in the gradient distance. Then, g_k monotone submodular.

Finally, the data summary is obtained by applying the greedy algorithm for the submodular cover problem, i.e. for a given $\epsilon_k \in]0,1[$, we look for the smallest possible subset $S_k \subseteq P_k$ such that $g_k(S_k) \geq (1-\epsilon_k)g_k(P_k)$. The greedy algorithm will then find a S_k^g such that $|S_k^g| \leq (1+\log(\max_e f(e)))\,|S^*|$, where S^* is the optimal subset.

It can be shown that $\|\nabla F_k(w) - \nabla F_{S_k}(w)\|$ is bounded with this method as is required in **A.4** for the convergence guarantee.

Lemma 1

$$\|\nabla F_k(w) - \nabla F_{S_k}(w)\| \leq \epsilon_k L_k(\{e_0\}) =: \sigma_k$$

.

Proof. From the definition of g_k and the fact that $g_k(S_k) \geq (1-\epsilon_k)g_k(P_k)$, we have

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

Thanks to our choice of e_0, $L_k(S \cup \{e_0\})$ is $L_k(S)$ for any set S, and the above inequality reduces to

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

which becomes

$$L_k(S_k) \leq \epsilon_k L_k(\{e_0\}),$$

since $L_k(P_k) = 0$.

The claim follows by substituting (7) in the above inequality. □

However, this method has the drawback that it 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 $\sum_{i \in P_k} |\nabla_w f_i(w) - \nabla_w f_{e_0}(w)\|$ under the previous constraint. This is in itself another optimization problem. Other works usually take the 0 point as their auxiliary element [14], however the theoretic results obtained with the optimal e_0 can't be verified in that case.

A second method for extracting S_k is to transform the problem in (5) into a monotone submodular one by putting it in the Facility Location form and to maximize:

$$\sum_{i \in P_k} \max_{j \in S_k} s(i,j) \tag{9}$$

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 [15], and it becomes the same for their application to DNNs. The greedy algorithm and its accelerated versions are equally applicable to this method. In the following, we will refer to Problem (9) mainly as the CRAIG problem. In the numerical experiments, we use the last-layer approximation as in CRAIG (see [15] for details).

4 Numerical experiments

All the experiments were performed using the Flower [2] Python package which provides an easy-to-use library for federated learning. Its source code is available in the GitHub repository [1]. 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. The dataset was either MNIST or CIFAR10.

In the legends, the terms 'ds' or 'craig' refer to the data summary obtained by solving Problem (9). The terms 'r' or 'rd' refer to random sampling. The term 'full' refers to using the full set. The number following the terms in the legend is the subset size.

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.

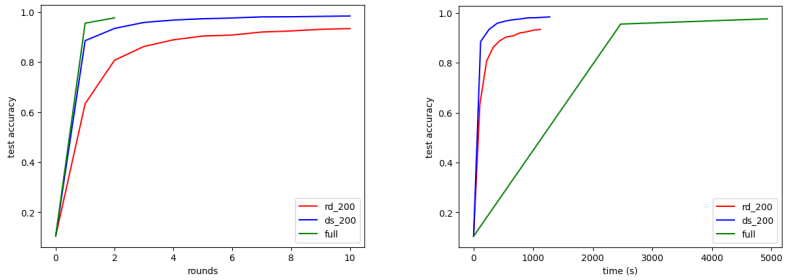
4.1 IID datasets

Our first tests were done using 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 [13] 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 Figs 2 and 3, the subset size is of 500 images while it is of 200 for Fig. 1. 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 train 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.

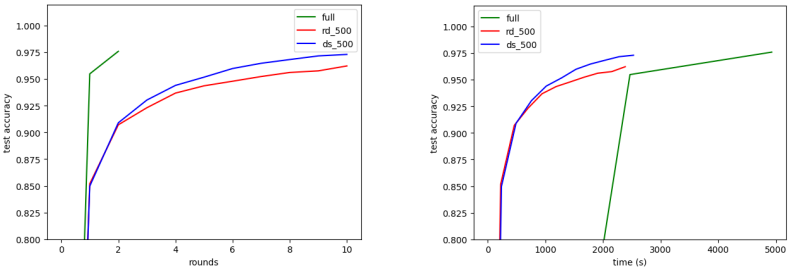
Figure 1 shows that using subsets greatly accelerates training in terms of execution time when compared to the full dataset in our setup. However, this requires more rounds. In that configuration, we were able to reach a test accuracy of 0.980 in 8 rounds taking 18 minutes with the data summaries while 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 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 much smaller compared to the whole training process.

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.



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

Fig. 1. 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 size is 0.0044.

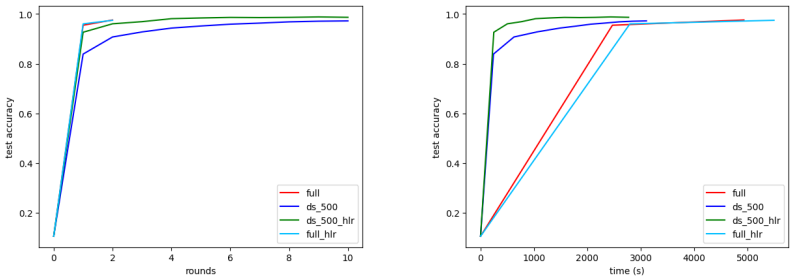


(a) Nb of epochs as x-axis. (b) Runtime as x-axis.

Fig. 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 0.011. FL setup.

Figure 3 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.

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



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

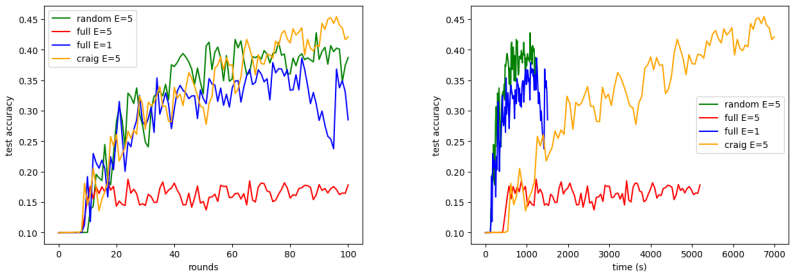
Fig. 3. Test accuracy compared between random sampling, CRAIG sampling or the full dataset for MNIST IID data. $E = 5$, $R = 10$, $\eta = 1e-2$ for the high learning rate (_hlr) and $\eta = 1e-3$ for the other. Full set size is 50000 and subset fraction size is 0.011.

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.

4.2 Non-IID datasets

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). As the computing time is reduced 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. [15][18][19].

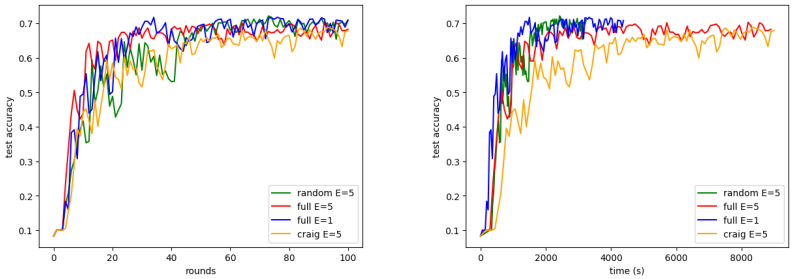
As shown in Figure 4, the configuration with 10 clients, using CRAIG for datasummary is far too time-consuming compared with random sampling or even using the full set, as mentioned earlier in this report. However, random sampling seems to perform better than using the full set, even if we reduce the 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



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

Fig. 4. 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.

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.



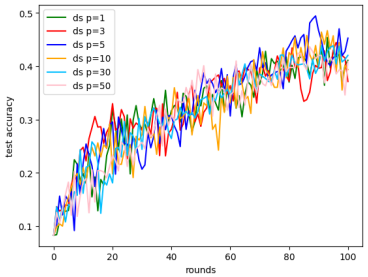
(a) Nb of rounds as x-axis. (b) Runtime as x-axis.

Fig. 5. 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.

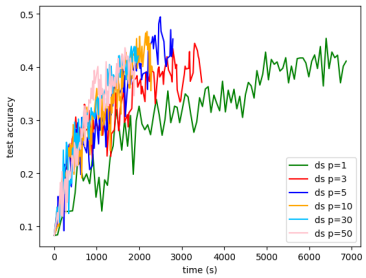
In the configuration with 4 clients as shown in Figure 5, the conclusion for the CRAIG data summarization is even clearer. As each local dataset here contains 12500 images, with 1250 in the validation set and the rest in the training set, extracting a subset containing 1250 images (a 0.11 fraction as 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 and 5 suggest that there are cases where using the full set even for only one epoch is not necessary, making the use of subsets relevant.

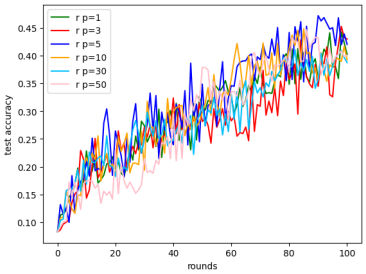
Frequency of subset selection In the previous section, using CRAIG was too time-consuming. One way of limiting this is to play on the frequency with which the subset is selected. On all our previous experiments, we selected a new subset at each round but we can also decide to choose a new subset each 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.



(a) Test accuracy per round for CRAIG sampling.



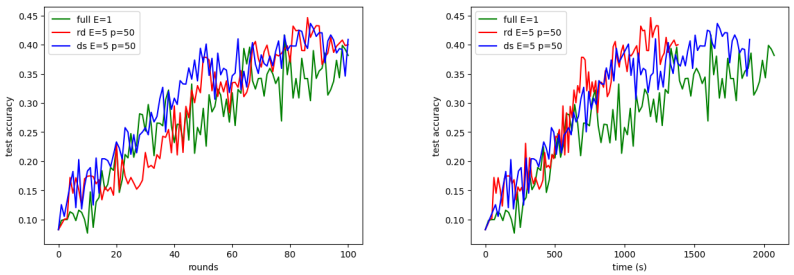
(b) Test accuracy as a function of time for CRAIG sampling.



(c) Test accuracy per round for random sampling.

Fig. 6. 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.

Figure 6 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.

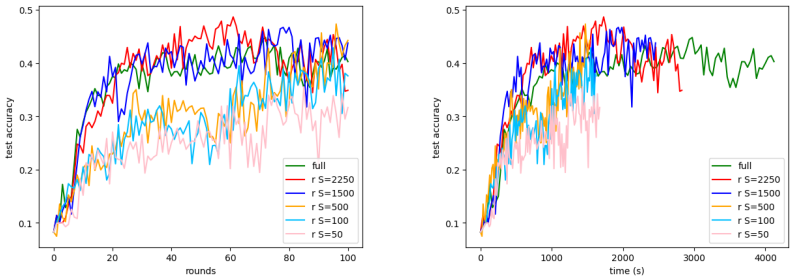


(a) Nb of rounds as x-axis. (b) Runtime as x-axis.

Fig. 7. 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.

Figure 7 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.

Size of subset Of course, the size of the selected subset is a hyperparameter that can greatly impact the training. Figure 8 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 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.



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

Fig. 8. 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.

5 Discussion and future work

Based on the results of our experiments, we can clearly identify two cases in which the use of subsets of local datasets seems relevant. In both however it is not clear how much improvement can be made by choosing the subset using a DS method rather than randomly selecting one.

The first case concerns clients with limited computational resources. In those cases, the training can take a very long time (the straggler problem) due to clients that are holding back other faster one. In some cases, it is important to also being able to include stragglers in the training loop, especially if the data points of the stragglers 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 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 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. Considering

the size of the subset, a balance has to be found as explained in part 4.2. 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.

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 and is part of our future work,

Another avenue of investigation is to determine the actual energy consumption of the algorithms with and without energy and determine the tradeoff between the accuracy and the energy consumption.

References

1. Flower github repository. *Available at https://github.com/adap/flower* (2023), `https://github.com/adap/flower`, last accessed 30 August 2023
2. Beutel, D.J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Kwing, H.L., Parcollet, T., Gusmão, P.P.d., Lane, N.D.: Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390 (2020)
3. Cui, L., Su, X., Zhou, Y., Zhang, L.: Clustergrad: Adaptive gradient compression by clustering in federated learning. In: 2020 IEEE Global Communications Conference, 2020, pp. 1-7 (2020)
4. Devillers, J.: Data summarization methods for energy efficient federated learning (October 2023), `https://github.com/juliannadvl/FL-DS/blob/main/rapport_stage.pdf`, Master's thesis internship report, ISAE-SUPAERO
5. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (jul 1998). https://doi.org/10.1145/285055.285059, `https://doi.org/10.1145/285055.285059`
6. Haddadpour, F., Kamani, M.M., Mokhtari, A., Mahdavi, M.: Federated learning with compression: Unified analysis and sharp guarantees. PMLR **130**, 2350–2358 (2021)
7. Jiang, P., Agrawal, G.: A linear speedup analysis of distributed deep learning with sparse and quantized communication. In: NeurIPS (2018)
8. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T.: Scaffold: Stochastic controlled averaging for federated learning. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning.

Proceedings of Machine Learning Research, vol. 119, pp. 5132–5143. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/karimireddy20a.html
9. Khaled, A., Mishchenko, K., Richtárik, P.: Better communication complexity for local SGD. CoRR **abs/1909.04746** (2019), http://arxiv.org/abs/1909.04746
10. Krause, A., Guestrin, C.: Near-optimal nonmyopic value of information in graphical models. In: Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence. p. 324–331. UAI'05, AUAI Press, Arlington, Virginia, USA (2005)
11. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks (2018). https://doi.org/10.48550/ARXIV.1812.06127, https://arxiv.org/abs/1812.06127
12. Li, X., Huang, K., Yang, W., Wang, S., Zhang, Z.: On the convergence of fedavg on non-iid data (2020)
13. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (20–22 Apr 2017), https://proceedings.mlr.press/v54/mcmahan17a.html
14. Mirzasoleiman, B.: Big Data Summarization Using Submodular Functions. Ph.D. thesis, ETH Zurich (2017)
15. Mirzasoleiman, B., Bilmes, J., Leskovec, J.: Coresets for data-efficient training of machine learning models. In: Proceedings of the 37th International Conference on Machine Learning. ICML'20, JMLR.org (2020)
16. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions - i. Mathematical Programming (1978)
17. Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.M., Rothchild, D., So, D., Texier, M., Dean, J.: Carbon emissions and large neural network training (2021)
18. Qiu, X., Parcollet, T., Fernandez-Marques, J., de Gusmao, P.P.B., Gao, Y., Beutel, D.J., Topal, T., Mathur, A., Lane, N.D.: A first look into the carbon footprint of federated learning (2023)
19. Reddi, S.J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., McMahan, H.B.: Adaptive federated optimization. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=LkFG3lB13U5
20. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green ai (2019)
21. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 3645–3650. Association for Computational Linguistics, Florence, Italy (Jul 2019). https://doi.org/10.18653/v1/P19-1355, https://aclanthology.org/P19-1355
22. Weinberg, M.: 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
23. Yu, H., Yang, S., Zhu, S.: Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning (2018)
24. Yuan, X.T., Li, P.: On convergence of fedprox: Local dissimilarity invariant bounds, non-smoothness and beyond (2022)