Aggregation Delayed Federated Learning

1st Ye Xue
Northwestern University
Evanston, USA
yexue2015@u.northwestern.edu

2nd Diego Klabjan
Northwestern University
Evanston, USA
d-klabjan@northwestern.edu

3rd Yuan Luo
Northwestern University
Chicago, USA
yuan.luo@northwestern.edu

***Abstract*—Federated learning is a distributed machine learning paradigm where multiple data owners (clients) collaboratively train one machine learning model while keeping data on their own devices. The heterogeneity of client datasets is one of the most important challenges of federated learning algorithms. Studies have found performance reduction with standard federated algorithms, such as FedAvg, on non-IID data. Many existing works on handling non-IID data adopt the same aggregation framework as FedAvg and focus on improving model updates either on the server side or on clients. In this work, we tackle this challenge in a different view by introducing redistribution rounds that delay the aggregation. We perform experiments on multiple tasks and show that the proposed framework significantly improves the performance on non-IID data.**

***Index Terms*—federated learning**

I. Introduction

As the amount of data generated by mobile devices increase explosively, followed by increasing privacy concerns of user data, researchers start seeking a solution to the dilemma of utilizing a large volume of user data while preserving the privacy of users. Federated learning is a machine learning paradigm that provides a solution to this dilemma. Under the coordination of a central server, a model is trained collaboratively by clients. To update the model, the server only collects a minimal amount of necessary information from clients but not their data [1]. Federated learning has been drawing increasing interest in recent years and has been applied in many on-device prediction tasks [2], [3]. The privacy promise of federated learning also makes it an appealing choice in healthcare applications [4], [5].

In federated learning, a global model is trained collaboratively on clients which are coordinated by a central server. Each round of training typically consists of four phases: an aggregation phase on the server, a local training phase on clients, and two communication (server-to-client and client-to-server) phases. The whole training process starts with a global model initialized on the server-side. In the server-to-client communication phase, a group of active clients is selected as training clients based on certain policy and the model is sent to them. Then, each client trains the model by calculating updates based on its own data stored on the local device. Stochastic gradient descent (SGD) is typically used to update local models during the local training phase. In the client-to-server communication phase, clients send their updated models back to the server, which then aggregates local models into a new global model in the aggregation phase.

Different from traditional distributed learning, in federated learning, client's raw data are never collected by the central server. This raises several challenges. First, data on a particular client are generated by a particular user, therefore, client data most likely are not distributed in a balanced and IID manner, which is usually an assumption in distributed learning. Second, the number of clients can be much larger than the number of samples on each client. This aggravates the issue of aggregation of non-IID clients. Third, clients are not always able to participate in training as user devices can be offline frequently or slow in communication. These challenges demand new methods different from existing algorithms designed for traditional distributed learning.

Our work focuses on mitigating the impact of non-IID client data distributions. Many existing works [6]–[10] adopt the FedAvg [1] framework and applied various strategies to handle non-IID data. In this work, we propose a new framework of federated learning with delayed aggregations. We delay the aggregation of local models on the server by redistributing local models to clients multiple times. Delayed aggregation and redistribution mitigate the non-IID issue by ensuring that each local model is trained with data of multiple clients before aggregation. In a non-IID scenario, a combination of data from multiple clients are potentially more representative of the global data distribution than the data of a single client.

Compared with several state-of-the-art federated learning algorithms that handle non-IID data distributions, our framework demonstrates a good ability to mitigate the impact of the non-IID data distribution and yields the best performance on multiple datasets. The main contributions are summarized as follows.

1) We propose a novel federated learning framework with delayed aggregations to handle non-IID data and a client selecting algorithm that further boosts the performance.
2) We evaluate the proposed algorithm on 9 non-IID datasets and demonstrate an improvement over the best benchmark by 1.56% on average. On multiple datasets with a more challenging task, our algorithm demonstrates an improvement of roughly 3% against the best comparison algorithm. The total wall-clock training time and the total communication requirements in our algorithm are the same as benchmarks. We implement our framework in Ray [11] with code made public [1].

[1]Code is available at: https://github.com/yxsub/RADFed

3) We propose a novel method to generate non-IID data. It samples non-IID sizes of clients, class distributions and feature distributions separately. This method provides an ability to simulate specific non-IID settings.
4) Along the way, we also study the impact of localized and global data standardization on federated learning.

II. Related Work

Many works have been done to tackle aforementioned challenges. Improving communication efficiency [1], [12], [13] is one of the most important topics in federated learning as client devices are usually on slow and expensive connections. Performing sketched updates is a popular strategy. Konečný et al. [12] applied quantization and subsampling on the model update to compress it before sending it back to the server. Wang et al. [13] reduce communication by avoiding irrelevant updates from clients. Each client determines if its update is relevant enough by checking whether its local update aligns with the global tendency.

Despite the great success of FedAvg, researchers showed that the performance of FedAvg reduces significantly when local data are non-IID [6]. Zhao et al. also proposed a strategy to mitigate non-IID data by sharing a subset of data between clients. The idea is to make the training data more IID through sharing. Many studies focus on handling the non-IID issue in this direction, [5], [14]. Instead of sharing raw data, a generative adversarial network (GAN) was trained in [14] to reproduce client data, which preserves privacy as no real data of clients is shared. A series of distillation methods was proposed in [15]–[17] to alleviate the non-IID issue through model distillation on extra data, such as public and synthetic data.

Another category of studies improving federated learning on non-IID data adds constraints when updating the model. This can be done either on clients or on the server side. Several works modified the loss function to correct or penalize local updates so that the local model is not shifted too much from the global model [8], [9], [18]. Wang et al. [19] proposed to normalize gradients on clients. Sattler et al. [10] proposed a communication-efficient federated learning framework to reduce communication costs by applying Top-k Sparsification. The sparsification restricts changes to a subset of the model's parameters and is shown to suffer the least from the non-IID data among existing model compression methods.

On the server side, Li et al. [7] applied momentum uniformly to the gradients of all clients to stabilize the training process under a non-IID scenario. However, collecting gradients from clients might require more frequent communications than collecting models from clients. Other momentum-based methods update the global model considering the historical global models [8], [20]. Reddi et al. [21] proposed adaptive federated learning algorithms, which treats the difference between the client's local update and the global model as pseudo-gradient and applied adaptive gradient descent algorithms to update the global model.

Our work focuses on handling non-IID data. Similar to [9], [21], we modify the FedAvg algorithm to make it more robust on non-IID data. Different from existing works, we change the aggregation logic by introducing redistribution rounds which delay the aggregation. Our method is orthogonal to aforementioned FedAvg-based methods and can be mixed with other algorithms. The proposed algorithm can be used as an alternative federated learning framework to FedAvg.

III. Methodology

One of the most common approaches to solve the optimization problem in federated learning is FedAvg [1]. In each training round, the server sends the global model to a subset of randomly selected clients. The clients update their local model using SGD on their own data in parallel and send back the updated model to the server. The server then updates the global model by averaging local updates from clients. Consider a subset $\mathcal{K}$ of training clients, the aggregation at the t-th round is written as

$$w_t \leftarrow \sum_{k \in \mathcal{K}} \frac{n_k}{n} w_t^k,$$

where w_t^k is the updated model on client k, n_k is the size of client k and n is the total size of clients. When data are identically distributed at clients, this aggregation works well since each local model is trained on a subset of data that is representative of the global distribution. It is identical to updating the global model in a centralized way. In non-IID cases, however, the client data can be highly skewed and it might not be a good idea to average the model trained on a highly skewed client with less skewed ones. The weighted averaging makes the aggregation even worse if a highly skewed client is large.

A. Delayed Aggregation

In order to make this aggregation work better in the non-IID setting, we have to answer the question: can each local model be trained on data that are representative of the global distribution at the time of aggregation?

One of the core promises federated learning makes is that no client data is collected by the server, so we can not make data on each client be representative of the global distribution by rearranging client data. However, we can rearrange local models. If we train a model on all clients one by one, we end up with a model that is trained on all the data. This would be similar to standard epoch based training and thus very slow. Second, it would assume that each client is active when needed. Alternatively, we can select only a subset of clients to perform this strategy.

Following this idea, we propose the Randomized Aggregation Delayed Federated learning algorithm (RADFed). We delay the aggregation by adding another training round to FedAvg. As shown in Algorithm 1, in the inner rounds, the server randomly sends local models back to clients again without performing aggregation. The server only aggregates local models at the end of the inner rounds. We call the inner

Algorithm 1 RADFed

K clients participate in training; C is the fraction of clients participating in each training round; T is the number of training iterations and S is the number of redistributing iterations.

Server executes:

initialize w_1
$m \leftarrow max(C \cdot K, 1)$
for each round $t = 1, 2, ..., T$ **do**
  $w_t^k \leftarrow w_t$, for $k = 1, 2, ..., m$
  $\bar{w}_1 \leftarrow (w_t^1, ..., w_t^m)$
  for each redistributing iteration $s = 1, 2, ..., S$ **do**
    $U \leftarrow$ uniformly sample m training clients
    for $i = 1, 2, ..., m$ **do**
      $\bar{w}_{s+1}^i \leftarrow$ ClientUpdate$(U_i, \bar{w}_s^i)$
    end for
  end for
  $w_{t+1} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \bar{w}_{S+1}^i$
end for
return w_{T+1}

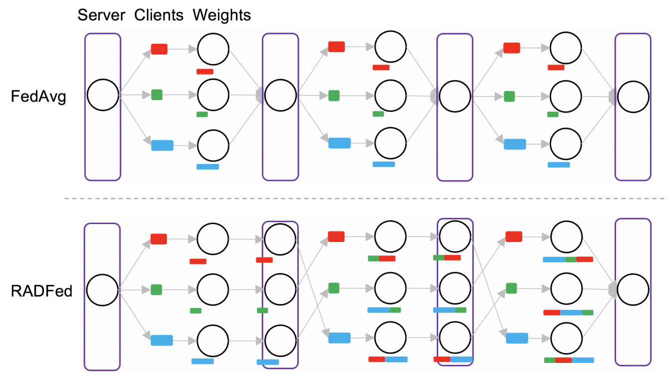


Fig. 1: A comparison of aggregation between RADFed and FedAvg. Colored blocks represent the data on clients. Each color represents a class. The length of each block shows the size of the data. Each circle represents a model's weights. Colored blocks under each circle show the data used to train a certain set of weights.

training rounds the redistributing rounds. ClientUpdate(k,w) trains the model of client k with initial weights w.

In Figure 1, we compare the aggregation in FedAvg and RADFed under a scenario with three clients, where each client has one class of data with a different number of samples. In FedAvg, there are three times of aggregation. At every aggregation, each set of local model weights is trained on data of a different class. In RADFed, however, there is one aggregation and two redistributions. At aggregation, each set of local model weights is trained on data of all three classes. In other words, at aggregation, local models in RADFed are trained with data that are more representative of the global distribution than in FedAvg. In this toy scenario, local models in RADFed are also trained with the same number of samples. In practice, this would not likely be the case, but they are expected to be trained with a similar number of samples due to randomization.

RADFed mitigates the non-IID issue with delayed aggregation and redistribution, where local models are trained on data that are more representative of the global data distribution. Note that, redistribution does not increase communication costs compared with FedAvg. Instead of redistribution, the server in FedAvg sends back the global model, which is the same size as local models. Additionally, as illustrated in Figure 1, with enough random redistributions, local models are expected to be trained on a similar number of samples. Therefore, we remove the sample size factor during aggregation and perform plain averaging over local models with equal weights. Because of this, the algorithm has another appealing property in terms of privacy-preserving in that the clients do not have to expose the size of their data. In many cases, the size of data can also be considered as sensitive information and exposing them may also cause privacy leakage. For example, it is more likely that a heavier user of a health-tracking app has a health problem. Furthermore, due to redistribution, the local updates sent by clients in RADFed are already aggregated information involving multiple clients thus hiding characteristics of each individual from attacks from untrustworthy parties. Although in the first redistribution round, the local update is not aggregated information, the client that receives this local update does not have the knowledge of whether this is aggregated information or not and does not have the knowledge of the source of the local update because it is sent from the server. In conclusion, the proposed method does not significantly increase, if not reduce, the privacy risk compared to FedAvg.

In practice, it is possible that the number of active clients is different in each round. To apply our framework, a small subset of active clients can be selected to make sure the number of active clients is the same across redistributing rounds. In some extreme cases where too few clients are active during redistribution, there are multiple strategies to make the framework work, e.g., reducing the number of redistributing models accordingly, or counting the number of times a local model has been redistributed and scheduling the redistributing process to make sure local models are redistributed a similar number of times before aggregation.

B. Importance Sampling

Not all samples are equally important and so are clients, especially in federated learning where client data are usually non-IID. If data are not identically distributed on clients, why should we select training clients through a simple uniform random sampling? We hypothesize that focusing computation on good clients can help improve federated learning algorithms. Inspired by [22], we propose RADFed-IS that incorporates the idea of importance sampling into our aggregation delayed framework. In [22] it is established that the optimal sampling probability is proportional to the square of the norm of the gradients.

The idea of importance sampling is to find a good mini-batch to train the model on in the next training step. A straightforward way of adopting this idea in our framework

Algorithm 2 RADFed+: RADFedp, RADFedm, RADFeda and RADFedi

Server executes:

initialize w_1
initialize p_k for each training client k
$m \leftarrow max(C \cdot K, 1)$
for each round $t = 1, 2, ..., T$ **do**
  $w_t^i \leftarrow w_t$, for $i = 1, 2, ..., m$
  $\bar{w}_1 \leftarrow (w_t^1, ..., w_t^m)$
  for each redistributing iteration $s = 1, 2, ..., S$ **do**
    $U \leftarrow m$ clients sampled with probabilities $\propto p$
    for $i = 1, 2, ..., m$ **do**
      $\bar{w}_{s+1}^i$, $p_{U_i}^{new} \leftarrow$ ClientUpdate$(U_i, \bar{w}_s^i)$
      $p_{U_i} \leftarrow (1-\gamma)p_{U_i} + \gamma p_{U_i}^{new}$
    end for
  end for
  $w_{t+1} \leftarrow$ $\alpha \cdot$ $\frac{1}{m}\sum_{i=1}^{m} \bar{w}_{S+1}^i$ $+(1-\alpha)w_t$
  $\Delta_t \leftarrow \frac{1}{m}\sum_{i=1}^{m}(\bar{w}_{S+1}^i - w_t)$
  $m_t, v_t \leftarrow$ Calculate momentum terms with respect to Δ_t, [21]
  $w_{t+1} \leftarrow w_t + \eta \frac{m_t}{v_t + \tau}$
end for
return w_{T+1}

ClientUpdate(k, w'):

On local data D, client k optimizes local objective function $F_k(w)$ or $F_k(w) + \frac{\rho}{2}||w - w'||^2$ starting with w'.
Let w^* be resulting solution.
$p = \frac{1}{|D|}\sum_{d \in D}||\nabla \ell_d(w^*)||_2^2$
return w^*, p to server

is to score the importance of all clients with respect to the current global model right after each aggregation and select the next set of clients to participate in training based on this score. However, collecting scores from all clients is usually not feasible in federated learning under the assumption that clients are not always active. Besides, it may increase the training time largely by adding an extra communication round to collect scores after each aggregation.

Instead, we score each client along with its local training. After local training, each client calculates the average square of the gradient norm of all mini-batches as its importance score and sends it back to the server along with the updated local model. The advantage of this strategy is that there is almost no extra burden added to the communication. Compared with the model itself, the size of an importance score can be neglected. However, the importance score calculated this way is no longer a good indicator of the importance of the client's data to the global model as each score is associated with a local model. In addition, a local model is not likely going to be trained on the same client in the next round because of the redistribution. Therefore, selecting clients based on this score might not be a good idea.

In order to solve this issue, we accumulate the importance scores for each client by averaging the scores calculated on all local models that have been trained on its local data. We expect that the accumulated score of a client becomes a good indicator of the importance of this client's data to all local models after accumulating over multiple rounds.

The server accumulates importance score p_k of client k by taking a weighted average between the old score and the new one as $p_k \leftarrow (1-\gamma)p_k + \gamma p_k^{new}$, with a mixing hyper-parameter $\gamma \in (0, 1)$. The server selects clients with probabilities proportional to the accumulated scores. Different from [23], we do not require to communicate raw gradients. The detailed algorithm RADFedi is shown in Algorithm 2.

Dataset	Min	Max	Mean	Stdev	C-score
Cifar10	2	2,850	600	605	1.29
Shakespeare	3	41,305	3,616	6,808	0.27
COVCLS	110	33,300	4,920	5,110	0.79
COVFEAT	372	17,328	4,920	3,237	0.68
MNIST-1	3	3,365	700	667	0.70
MNIST-0.1	11	3,327	700	658	1.29
eICU	108	5,683	901	925	0.06

TABLE I: Statistics of datasets (number of samples of clients)

C. RADFed+

RADFed itself works independently as a federated learning algorithm. However, it can also be used as a federated learning paradigm, as an alternative to FedAvg, where other methods can be plugged in. In this work, we propose several RADFed-based algorithms RADFed+ under this paradigm, including RADFed+FedProx (RADFedp), RADFed+Momentum (RADFedm), RADFed+Adaptive (RADFeda) and RADFed+IS (RADFedi). These algorithms cover methods designed to improve the following major aspects in federated learning: server-side updates, client-side updates and client sampling. Although we only select one or two methods from each aspect in this experiment, other similar methods can be applied on top of RADFed as well. The algorithm of RADFed+ is shown in Algorithm 2.

IV. Experimental Setup

In this work, we focus on evaluating the performance of federated learning algorithms in non-IID settings. Although a real-world non-IID dataset is ideal, datasets with an artificial partition are also very helpful in simulating different non-IID settings. Many studies create heterogeneous clients by manually sampling data on clients so that the class distribution is not identical across clients. In existing sampling methods, the sizes of clients are usually determined by class sampling. To the best of our knowledge, feature-imbalance has not been considered in prior works.

In order to simulate non-IID settings with more control of the distribution of sizes, classes and features, we propose a sampling method where we can sample them independently with a different Dirichlet prior. It is not always the case that we can draw a desired number of samples to satisfy all these independently sampled distributions at the same time. Let us consider sampling non-IID sample sizes and classes as an example. A sampling solution for T clients and C classes is a $T \times C$ matrix where each entry denotes the number of samples of class c on client t. By sampling sizes and classes separately, we specify each entry of the matrix, that is a total of $T \cdot C$ numbers. However, given the number of samples in each class in a dataset, we only need $T \cdot C - C$ entries to specify a solution. Therefore, we propose a Quadratic Programming (QP) method to find a random feasible sampling solution.

Algorithm 3 Random QP solution

Input: QP solution $A = \{\alpha_{tk}\}_{tk}$

for $p = 1, 2, ..., P$ **do**
  $A \leftarrow$ RandomizeSolution(A) // a burn-in period
end for
$h \leftarrow \infty$
for $q = 1, 2, ..., Q$ **do**
  $A \leftarrow$ RandomizeSolution(A)
  $L(A) \leftarrow$ calculate loss from (1) or (2)
  if $L(A) < h$ **then**
    $\bar{A} \leftarrow A$
    $h \leftarrow L(A)$
  end if
end for
return $\bar{A}$

RandomizeSolution(A):

$(i, j), (\bar{i}, \bar{j}) \leftarrow$ indices of two random entries of A
$\varepsilon \leftarrow$ uniform$(0, \min\{A_{i,j}, A_{\bar{i},\bar{j}}, \xi\})$
for Each position (m, n) **do**

$$A^{new}_{m,n} \leftarrow \begin{cases} A_{m,n} - \varepsilon, m = i, n = j \\ A_{m,n} - \varepsilon, m = \bar{i}, n = \bar{j} \\ A_{m,n} + \varepsilon, m = i, n = \bar{j} \\ A_{m,n} + \varepsilon, m = \bar{i}, n = j \\ A_{i,j}, \text{otherwise} \end{cases}$$

end for
return A^{new}

A. Partitioning of Heterogeneous Data

Let C_k be the number of samples of class k and N be the total number of samples. Clearly, we have $N = \sum_k C_k$. Let $n \sim Dir(\mu)$ be the sizes of clients and $c_t \sim Dir(\lambda_t)$ be the class distribution of client t. Let α_{tk} be the number of samples of client t of class k. We want $\alpha_{tk} = c_{tk} n_t N$, given n_t and c_{tk}. However, the dataset needs $\sum_t \alpha_{tk} = C_k$ and $\sum_k \alpha_{tk} = n_t N$, which they might not hold. Therefore, we find a feasible solution for α by solving

$$\min_{\alpha \geq 0} \sum_{t,k} (\alpha_{tk} - c_{tk} n_t N)^2 \tag{1}$$

subject to $\sum_k \alpha_{tk} = n_t N, \forall$ client t and $\sum_t \alpha_{tk} = C_k, \forall$ class k, which is a convex QP.

Using the similar idea of sampling classes and sizes, we also sample categorical features in a non-IID manner. We sample category distribution $f^j_t \sim$ Dir(θ^j_t) of feature j with d_j categories on client t. We consider classes as a feature that are sampled separately. Let U be a set of all possible combinations of categories in the dataset and B_u be the number of samples that fall into configuration $u \in U$. The first element of u corresponds to classes. Now let α_{tu} be the number of samples on client t with configuration u. Then we find a feasible solution through

$$\begin{aligned} \min_{\alpha \geq 0} \sum_t [\sum_{k=1}^{K} (\sum_{u \in U, u_1 = k} \alpha_{tu} - c_{tk} n_t N)^2 \\ + \sum_{j=1}^{M} \sum_{i=1}^{d_j} (\sum_{u \in U, u_{j+1} = i} \alpha_{tu} - f^j_{ti} n_t N)^2] \end{aligned} \tag{2}$$

subject to $\sum_{u \in U} \alpha_{tu} = n_t N, \forall$ client t and $\sum_t \alpha_{tu} = B_u, \forall$ configuration $u \in U$. Here M is the number of categorical features. If a feature is non-categorical by nature, we can create buckets that correspond to categories.

The above QPs may have many optimal solutions but we want a random one. We generate a random solution by modifying values at the "4 vertices of a random rectangle," in a way that the modified values still satisfy constraints in (1) or (2). Algorithm 3 finds a random QP solution. A step size ξ is used to control the modification. The algorithm has two phases. In the first phase, we find a suboptimal solution by randomly modifying values. Then, in the second phase, starting from the suboptimal solution, we continue modifying and record the best random solution we find.

B. Datasets and Models

For all datasets that are partitioned by (1) or (2), we set $\mu = 1$. These datasets have reasonably large variations in client sizes, see Table I. We use the same $\lambda = 0.1$ for all clients, on all datasets but MNIST, where we experiment with different values of λ. Other hyper-parameters used in this paper are specified in IV-D. For evaluation, we split clients into training/validation/test groups [2], [5] and perform 5-fold cross-validation to reduce the selection bias, which might be aggravated by the non-IID client distributions, see details in IV-C.

Covertype is a large structured dataset for forest cover type prediction from the UCI KDD archive [24]. We partition the

Dataset	RADFed	RADFed-IS	
	S	S	γ
COVCLS(-L/-G)	22	22	0.9
COVFEAT(-L/-G)	20	20	0.9
MNIST-1	22	22	0.9
MNIST-0.1	15	15	0.9
Cifar10	15	100	0.9
Shakespeare	15	100	0.8
eICU	80	80	0.9

TABLE II: Hyper-parameters in proposed algorithms

Dataset	FedAvg	FedProx	FedAsync	FedAdapt	RADFed	RADFedi	RADFeda	RADFedp	RADFedm
Cifar10	(82.88)	0.84	2.11	0.39	2.74	**3.03**	2.20	3.00	2.28
Shakespeare	(51.71)	0.33	0.75	0.54	4.04	**4.51**	3.04	3.83	3.96
COVFEAT-G	(87.61)	0.37	0.34	−1.45	3.23	2.59	2.49	**3.37**	1.96
COVFEAT-L	(78.79)	1.29	0.93	−1.50	5.01	4.28	3.96	4.78	**5.29**
COVCLS-G	(93.87)	0.03	−0.20	−0.26	0.16	0.23	0.26	0.26	**0.33**
COVCLS-L	(90.62)	0.38	0.06	−0.23	2.27	0.09	2.24	2.16	**2.32**
MNIST-1	(97.29)	0.13	−0.05	0.11	0.16	0.23	0.23	0.41	**0.47**
MNIST-0.1	(96.95)	0.15	−0.17	0.10	0.24	0.29	0.36	0.20	**0.42**
eICU	(92.21)	0.00	0.11	0.05	0.11	0.08	**0.13**	0.11	0.11
AVG	(85.77)	0.37	0.38	−0.26	1.79	1.47	1.51	**1.82**	1.71

TABLE III: Average test performance of 5-fold cross-validation: % accuracy for the MNIST, Cifar10 and Shakespeare dataset; F1 score ($\times 100$) for all Covertype datasets; and Area Under the Receiver Operating Characteristic Curve (AUC) ($\times 100$) for the eICU dataset. The absolute scores are reported for FedAvg and the % relative performance difference against FedAvg is shown for other algorithms.

data into 100 clients. The number of training clients (K) is 60. The number of validation and test clients are 20 each. Same sizes are used for the MNIST and Cifar10 datasets. We train a fully connected neural network with 2 hidden layers with 64 neurons each.

We create two types of datasets, one (COVCLS) with classes and client sizes sampled non-identically based on (1) and the other (COVFEAT) with also features sampled non-identically thus using (2). For both datasets, we set $\lambda = 0.1$ for all clients, and set $\theta = 0.1$ for the COVFEAT dataset. All the datasets that follow are created based on (1).

On the Covertype datasets, we also study the impact of localized and global data standardization. The difference is whether to use global statistics of all clients' data to standardize client local data or to let each client perform standardization with its own statistics. On COVFEAT-G and COVCLS-G, we perform global standardization, while on COVFEAT-L and COVCLS-L, localized standardization is used. When comparing our algorithm with benchmarks on other datasets, we use global standardization to be consistent with the original papers.

MNIST [25] consists of images of digits with 10 classes. We sample 100 clients with classes and sizes non-identically distributed. We study how data heterogeneity impacts the performance of federated learning algorithms by creating two datasets, MNIST-1 with $\lambda = 1$ and MNIST-0.1 with $\lambda = 0.1$. A dataset generated with the larger λ has a lower heterogeneity in class distributions. We build a fully connected neural network same as [1].

Cifar10 [26] images are partitioned into 100 clients with classes ($\lambda = 0.1$) and sizes ($\mu = 1$) non identically distributed. We use pre-trained MobileNetV2 [27] as the model and train a subset of layers from the last bottleneck convolution layer to the classification layer.

Shakespeare dataset is a language modeling dataset built from *The Complete Works of William Shakespeare* [1]. We use the same data as [9] but partition samples by speaking roles. Each speaking role corresponds to one client. In total, the dataset consists of 143 clients. The number of training, validation and test clients are $85, 29$ and 29, respectively. The task is to predict the next character given a sequence of 80 characters. We train a 2 layer long short-term memory (LSTM) classifier with an 8-dimensional embedding layer.

The **eICU** critical care database is a large multi-center database made available by Philips Healthcare [28]. We train a logistic regression model to predict the in-hospital mortality using variables underlying the Acute Physiology Age Chronic Health Evaluation (APACHE) predictions [2]. To avoid a potential sampling bias, we focus on mid to large hospitals with more than 100 admissions and exclude those associated with a high mortality rate (greater than 20%). Each hospital corresponds to a client. The dataset contains 164 clients. The number of training, validation and test clients are $98, 33$ and 33, respectively.

C. Experimental Setup

To our best knowledge, there is no gold standard for evaluating federated algorithms. Generally, there are 3 ways to split the data into training and test sets: splitting all data globally [4], [7], [10], splitting each client's local data [3], [21], [29] and splitting clients into training/test groups [2], [5]. In this work, we adopt the last strategy by assuming no local data can be collected by the server and the server can not manipulate the client's local data. Additionally, we perform 5-fold cross-validation with the by-client splits in order to reduce the selection bias, which might be aggravated by the non-IID client distributions. We split all clients into 5 sets. One by one, a set is selected as the test set. For the remaining sets, one by one, a set is selected as the validation set and the others are used as the training set.

D. Hyper-parameters

For feature sampling, we set $\theta = 0.1$ for all clients and features. In QP, we use $P = 10^5$, $Q = 5 \cdot 10^5$ and $\xi = 0.002$. The impact of C, B (the mini-batch size) and E (the number of local training epochs) is well studied and thus we do not focus on experimenting on various settings of these variables. We set

[2]The full variable list and descriptions are available at https://eicu-crd.mit.edu/eicutables/apachepredvar and https://eicu-crd.mit.edu/eicutables/apacheapsvar/.

$C = 0.1$, which is shown to be a generally good setting that balances the performance and the convergence speed [1]. The mini-batch size B is set to 10 and 16 for MNIST and Cifar10, respectively, considering that clients on these datasets do not have many samples. On other datasets, B is set to 256. We set $E = 10$ for MNIST to make the task more challenging and set $E = 1$ for the other datasets. Besides these general federated learning hyper-parameters as mentioned above, each particular algorithm has its own hyper-parameters. RADFed has one more hyper-parameter, the number of redistribution rounds S, than FedAvg. RADFedi adds another hyper-parameter, the mixing weight γ. We tune hyper-parameters specified by each federated learning algorithm using grid search on validation clients. The number of redistribution rounds S and the mixing weight γ are tuned on $\{10, 15, 20, 22, 25, 28, 30, 50, 80, 100, 120\}$ and $\{0.6, 0.7, 0.8, 0.9, 0.95\}$, respectively. Table II lists the hyper-parameter values of proposed algorithms used in our experiments. For other RADFed+ algorithms, we use the best hyper-parameters found in RADFed and the other corresponding method. Note that, the values of hyper-parameters in RADFedm are from FedAsync, where the global model is updated in the same way as RADFedm. Hyper-parameters used in comparison models are detailed in the next section.

E. Comparison Models

We compare the performance of our method RADFed with FedAvg [1], FedProx [9], the adaptive federated operation method (FedAdapt) [21], the asynchronous federated optimization method (FedAsync) [8] and RADFed+ on multiple tasks. FedAvg is probably the most popular and commonly used federated algorithm and the others are the state-of-the-art federated learning algorithms that handle non-IID data distributions. FedAsync is an asynchronous method whose performance can be impacted largely by staleness. We delicately set its value to make a fair comparison between asynchronous and synchronous methods. We add FedAsync, which is an asynchronous algorithm, as one of the benchmarks to make a more comprehensive comparison regarding the non-IID problem, as it has several techniques to handle this issue.

Different from synchronous methods, FedAsync has to deal with the staleness of updates from clients. The staleness of a client's update is defined as the timestamp difference between a client's update and the server's model. The performance of FedAsync suffers from large staleness. In order to mitigate the impact of staleness on training, the new global model is updated as a weighted average between the old global model and the client's local update. In addition, the authors show that decaying the mixing weights as a function of staleness helps to fight against large staleness. Despite these efforts, the impact of staleness on FedAsync's performance is not completely eliminated.

In order to make a fairer comparison between asynchronous and synchronous methods, we have to choose a reasonable value for staleness. We simulate the FedAsync's training procedure and find maximum staleness where the average number of clients running in parallel per round is the same as in the synchronous methods. In other words, we compare the performance of FedAsync and synchronous methods under the same level of parallelism on average. The $maximum\ staleness$ defined in [8] is set to 18, 19, 18, 28 and 31 for MNIST, Cifar10, Covertype, Shakespeare and eICU datasets, respectively.

We tune hyper-parameters of the benchmark algorithms by grid search and select the best configuration based on the performance on validation clients. We tune μ of FedProx [9] on values $\{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}\}$. For FedAdapt, we fix the momentum terms of 0.9 and $\beta_2 = 0.99$ throughout for all optimizers as suggested in [21]. We then tune η, τ and v_{-1} on sets $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, $\{10^{-2}, 10^{-4}, 10^{-8}\}$ and $\{0, 10^{-1}, 10^{-2}, 10^{-3}\}$, respectively. We tune ρ and α of FedAsync on $\{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}\}$ and $\{0.3, 0.6, 0.9\}$, respectively. For FedAsync with adaptive mixing hyperparameters, we take $a = 10$ and $b = 4$ for FedAsync+$Hinge$ and $a = 0.5$ for FedAsync+$Poly$ as suggested in [8].

V. Results

We run each algorithm 3 times with different seeds on each of the 5 folds and report the average performance over the 15 runs in Table III. On average, RADFed and the best RADFed+ model offer an improvement over FedAvg by 1.79% and 1.82%, respectively, and are 1.40% and 1.44% better than the best benchmark (FedAsync), respectively. On the MNIST datasets and the eICU dataset, all algorithms achieve a close performance. On other datasets, the RADFed-based models are significantly better than FedAvg and the best benchmark ($p < 0.05$ under the Wilcoxon signed-rank test [30]). Under some difficult settings, which we discuss later, our framework offers a substantial improvement over all comparison algorithms on multiple datasets. The best of RADFed-based models outperforms the best benchmark by 3.72% on Shakespeare, 3.01% on COVFEAT-G, 4.33% on COVFEAT-L and 2.26% on COVCLS-L.

Figure 2 shows that RADFed is quite stable across different seeds and confirms its significant improvement on these datasets. In Figures 3 and 4, we compare the validation curves. On each dataset, the best variant of RADFed+ is shown. With delayed aggregations, RADFed and RADFed+ stabilize the training by demonstrating a smaller variation in validation scores than the algorithms that adapt the FedAvg framework. Note that the x-axis of each plot is the number of local training round. The number of aggregations in RADFed-based algorithms is $1/S$ of the number in FedAvg-based algorithms. The total wall-clock training time is the same with the same number of local training rounds, assuming the local training time at clients is the same. In general, our algorithms achieve the maximum validation score at a similar number of training rounds as other algorithms. On Shakespeare, our algorithms peak much later than FedAvg. This is due to a large learning rate used in FedAvg, where a relatively larger learning rate yields a better result, although the model gets overfitted quicker than using a lower learning rate. It does not imply

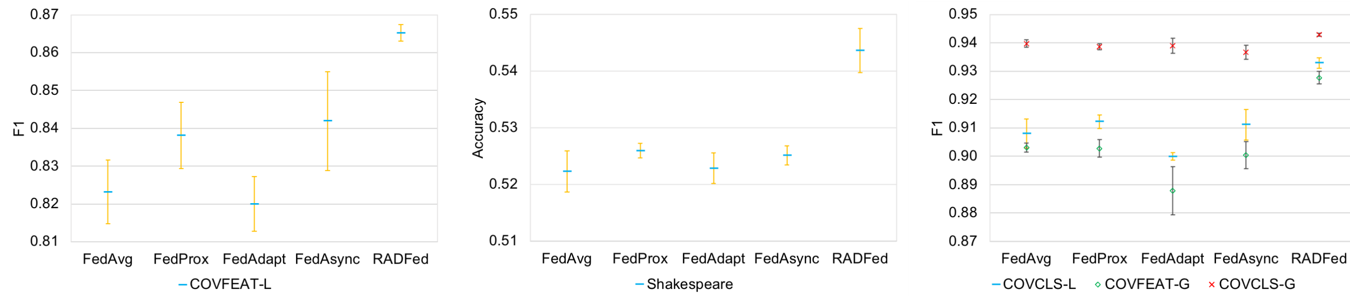


Fig. 2: Test performance comparison on the Covertype and Shakespeare datasets. Multiple runs are performed with different seeds on the most representative fold, defined as the one with the closest performance gap to the average of all folds. The performance gap is the difference in the test performance between RADFed and FedAvg.

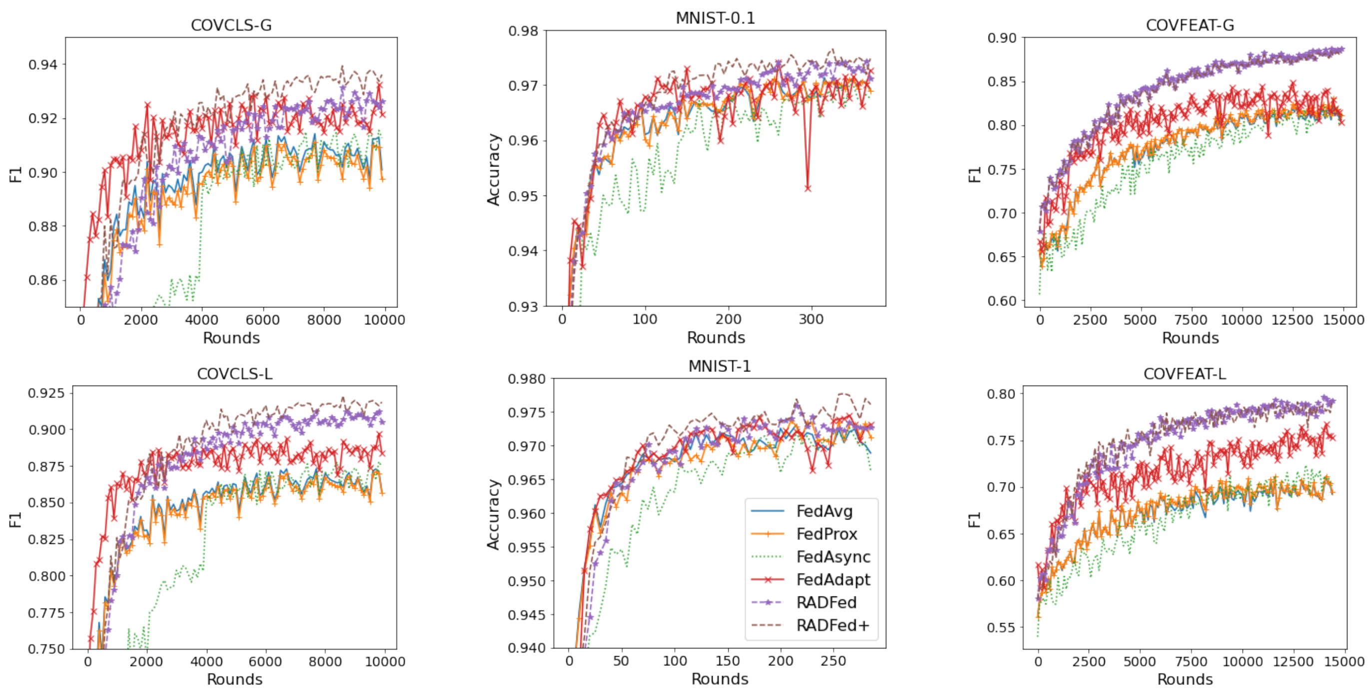


Fig. 3: Validation performance comparison on the Covertype and MNIST datasets. The F1 score is used on all Covertype datasets and accuracy is reported on the MNIST datasets. The x-axis is the number of training rounds, which is the count of regular training rounds in FedAvg-based algorithms and the count of redistribution rounds or inner rounds in RADFed-based algorithms. Curves are smoothed by taking the average over evenly spaced intervals for better visualization. The intervals are chosen differently considering that validation frequencies are different. The intervals are set to 100 for the Covertype datasets and 5 for the MNIST datasets.

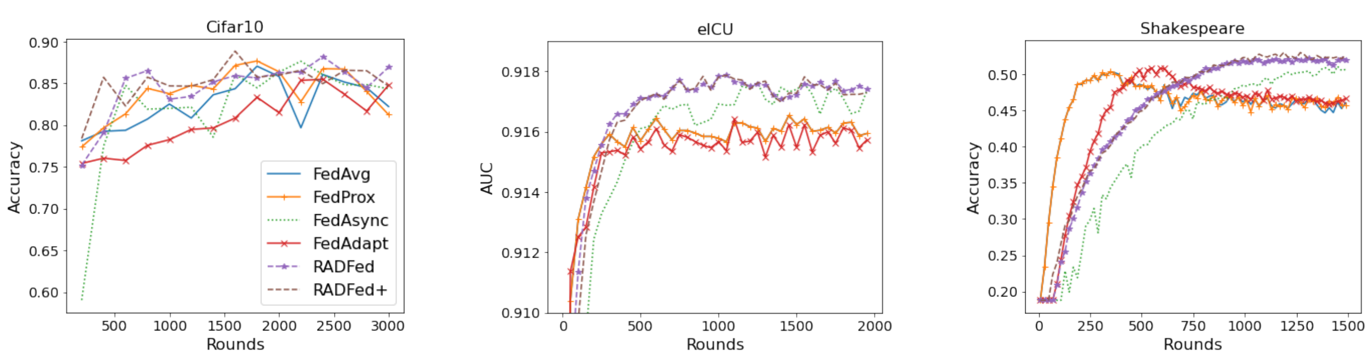


Fig. 4: Validation performance comparison. Accuracy is reported on the Cifar10 and Shakespeare datasets. AUC is used on the eICU dataset. Similar to Figure 3, curves are smoothed with intervals 1, 2 and 50 for the Cifar10, Shakespeare and eICU datasets, respectively.

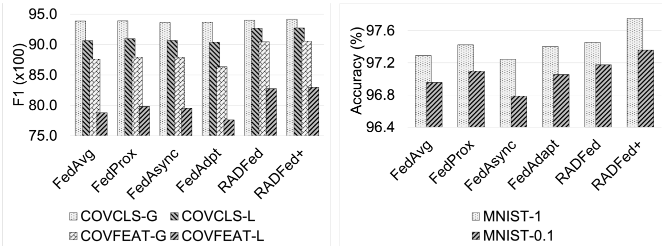


Fig. 5: Comparison on different levels of heterogeneity

that delaying aggregations also delays convergence. In fact, on Shakespeare, aggregations in RADFed are delayed with 15 redistribution rounds and the number in RADFedi is 100. We observe a similar convergence behavior.

A. Performance of RADFed+

On each dataset, the best performance is achieved by one of the RADFed+ models. RADFedi is the best on Cifar-10 and Shakespeare. RADFedp is the best on one of the Covertype datasets (COVFEAT-G). RADFedm is the best on the other three Covertype and both MNIST datasets. RADFeda is the best on the eICU dataset. The results demonstrate that our algorithm can work together with many other algorithms that improve federated learning in various aspects, including server-side changes, client-side modifications and client sampling. We also observe that RADFed based models are better than their FedAvg-based variants: RADFeda and RADFedp show significant improvements on all datasets (up to 3.5% and 5.5%) when compared to their FedAvg-based variants FedAdapt and FedProx, respectively.

B. Divergence on Delayed Updates

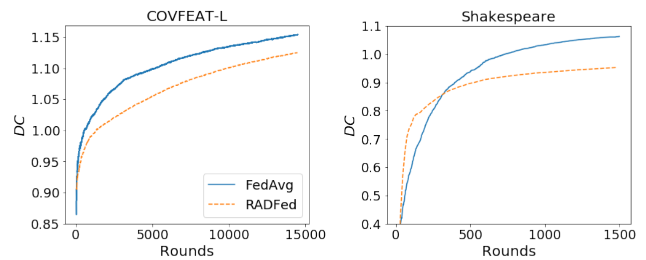
In studying the non-IID challenge in federated learning, the weight **D**ivergence between the **C**entralized and federated models (DC) has been used to explain the performance reduction, which as shown in [6] can be attributed to the divergence. It is defined as $DC(t) = \frac{||\mathbf{w}_{FL}^t - \mathbf{w}_C^t||}{||\mathbf{w}_C^t||}$, where $\mathbf{w}_{FL}^t$ are the weights of the global model in federated training at the t-th round and $\mathbf{w}_C^t$ are the centralized weights.

To visualize the weight divergence DC, we train a centralized model and a federated model side by side. Both models start with the same weight initialization. In each round, the same data are used in training. The difference is that in centralized training we collect data from clients and update the model using combined data. The divergence from the centralized model is expected due to the distance between the client data distribution and the population distribution. As shown in Figures 6a and 6b, RADFed algorithm demonstrates a smaller weight divergence than FedAvg. It indicates that the aggregated weights of our algorithm are less impacted by the skewness of the data and are closer to the weights trained on data under the population distribution.

We also visualize the **D**ivergence between clients' **L**ocal updates (DL), which helps understand how our algorithm behaves. For a set of clients $\mathcal{K} = \{1, 2, ..., K\}$, the divergence is defined as

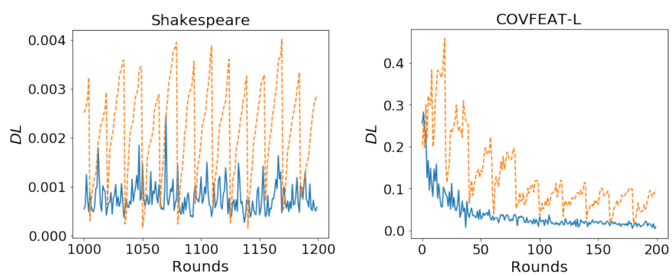
$$DL(\mathbf{w}_1^t, \mathbf{w}_2^t, ...\mathbf{w}_K^t) = \binom{K}{2}^{-1} \sum_{i,j \in \mathcal{K}; i<j} (1 - \frac{\mathbf{w}_i^t \cdot \mathbf{w}_j^t}{\|\mathbf{w}_i^t\| \, \|\mathbf{w}_j^t\|}), \quad (3)$$

where $\mathbf{w}_i^t$ is the local update from client i in the t-th round. A positive correlation between DL and federated learning performance is observed in [31]. The study is based on the FedAvg framework that is different from ours. Although the same correlation might not hold when comparing different frameworks, it helps visualize how our algorithm behaves.



(a) (b)

Fig. 6: Weight divergence



(a) (b)

Fig. 7: Weight divergence among local updates

During training of our algorithm, we observe a periodical trajectory of DL, Figure 7. In the first round after each aggregation, the divergence is the smallest. As the aggregation being delayed for more rounds, the divergence keeps increasing until the next aggregation. The divergence in FedAvg vibrates around the lowest values of our algorithm. Figures 7a and 7b show the weight divergence of local updates on Shakespeare and COVFEAT-L datasets.

The increasing DL does not indicate any deficiency of our framework. It might be due to the nature of the redistribution of local models. For example, in a non-IID setting where each client has one class of data, training may start with clients of different classes and yield large divergence between local models. In the next redistribution round, the divergent local models are trained again on client data of different classes. The divergence accumulates as the redistribution continues. FedAvg, however, results in a smaller DL because it performs aggregation after each local training and divergence is not accumulated.

C. Heterogeneity

We create datasets with various levels of heterogeneity to evaluate whether our model is effective and robust under different heterogeneous settings. In order to compare between manually partitioned datasets and naturally partitioned ones, we introduce the class non-IID score (C-score), which is defined as $\frac{1}{K}\sum_{k=1}^{K}\sum_{c=1}^{C}|r_c^k - R_c|$, where r_c^i is the ratio of class c on client k and R_c is the ratio of class c in all data. This score measures the difference between client's class ratios and the global class ratios. C-score of each dataset is shown in Table I. For example, The MNIST-0.1 dataset is expected to have a higher heterogeneity of class distributions than the other due to a smaller value of λ, and its C-score is higher than in MNIST-1. The COVCLS and COVFEAT datasets are partitioned with the same value of μ and λ, so they have a similar level of heterogeneity with respect to client sizes and classes. Their C-score are also similar. However, since we also introduce heterogeneity on feature distributions in the COVFEAT datasets, they should have a severer issue on non-IID data distribution than COVCLS datasets.

In the heterogeneity experiment, we observe that all algorithms perform worse on MNIST-0.1 than MNIST-1 and all algorithms show a lower performance on COVFEAT datasets than on COVCLS datasets, no matter which standardization method is used, as shown in Figure 5. RADFed and RADFed+ outperform benchmarks on all these datasets of different levels of heterogeneity, which demonstrates the robustness of proposed algorithms in various heterogeneous settings.

D. Standardization

With global standardization, RADFed outperforms FedAvg by 0.17% and 3.24% on COVCLS-G and COVFEAT-G, respectively. With localized standardization, we observe a performance regression on all federated learning algorithms. However, RADFed demonstrates a good ability in handling localized standardization by offering a larger performance improvement over FedAvg on both COVCLS-L (2.3%) and COVFEAT-L (5.0%).

Interestingly, RADFed outperforms RADFedi on both COVCLS-L and COVFEAT-L and RADFedi ranks 4th and 5th in all 5 RADFed-based models, which implies that it is more challenging for RADFedi to determine which clients are better under localized standardization. RADFedi works well under global standardization. It improves RADFed on 5 datasets and ranks the best on 2 datasets.

VI. Conclusion

In this work, we propose a new training framework with delayed aggregation to handle the well-known non-IID issue in federated learning. We demonstrate that our framework offers a substantial improvement over the FedAvg framework and outperforms several state-of-the-art federated learning algorithms. Moreover, we incorporate in our framework importance sampling and several other techniques improving federated learning in various aspects and further improve the framework on multiple datasets.

Along the way, we also discuss the following topics in federated learning: the splitting of training and test sets, localized and global standardization, and weight divergence on different frameworks. Experiments show that federated learning algorithms suffer from localized standardization. The proposed framework demonstrates a good ability in handling localized standardization. However, the importance sampling version does not offer further improvement under localized standardization. In addition, we propose a sampling algorithm to generate non-IID datasets. It offers the choice for a desired non-IID level on client sizes, classes and features separately, thus providing researchers with more flexibility and control about simulating different non-IID settings.

Acknowledgment

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[2] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," 2019.

[3] E. Bakopoulou, B. Tillman, and A. Markopoulou, "Fedpacket: A federated learning approach to mobile packet classification," *IEEE Transactions on Mobile Computing*, 2021.

[4] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *Journal of biomedical informatics*, vol. 99, p. 103291, 2019.

[5] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, "Loadaboost: Loss-based adaboost federated machine learning with reduced computational complexity on iid and non-iid intensive care data," *Plos one*, vol. 15, no. 4, p. e0230706, 2020.

[6] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018.

[7] C. Li, R. Li, H. Wang, Y. Li, P. Zhou, S. Guo, and K. Li, "Gradient scheduling with global momentum for non-iid data distributed asynchronous training," 2019.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2020.

[9] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems*, 2020.

[10] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[11] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," in *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 561–577.

[12] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017.

[13] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems*, 2019, pp. 954–964.

[14] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," 2018.

[15] D. Yao, W. Pan, Y. Dai, Y. Wan, X. Ding, H. Jin, Z. Xu, and L. Sun, "Local-global knowledge distillation in heterogeneous federated learning with non-iid data," *arXiv preprint arXiv:2107.00051*, 2021.

[16] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and

H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/18df51b97ccd68128e994804f3eccc87-Abstract.html

[17] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," *arXiv preprint arXiv:1910.03581*, 2019.

[18] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: stochastic controlled averaging for federated learning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 5132–5143. [Online]. Available: http://proceedings.mlr.press/v119/karimireddy20a.html

[19] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[20] T. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *CoRR*, vol. abs/1909.06335, 2019. [Online]. Available: http://arxiv.org/abs/1909.06335

[21] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *9th International Conference on Learning Representations*, 2021.

[22] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *International conference on machine learning*, 2018, pp. 2525–2534.

[23] E. Rizk, S. Vlaski, and A. H. Sayed, "Federated learning under importance sampling," *CoRR*, vol. abs/2012.07383, 2020. [Online]. Available: https://arxiv.org/abs/2012.07383

[24] S. Hettich and S. D. Bay, "Uci kdd archive," http://kdd.ics.uci.edu, 1999.

[25] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Technical Report, 2009.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[28] T. J. Pollard, A. E. Johnson, J. D. Raffa, L. A. Celi, R. G. Mark, and O. Badawi, "The eicu collaborative research database, a freely available multi-center database for critical care research," *Scientific data*, vol. 5, no. 1, pp. 1–13, 2018.

[29] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," 2019.

[30] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[31] H. Kim, T. Kim, and C.-H. Youn, "On federated learning of deep networks from non-iid data: Parameter divergence and the effects of hyperparametric methods," 2020. [Online]. Available: https://openreview.net/forum?id=SJeOAJStwB