Client-side clustering algorithms

Dr. Tanghatari Research Group

Topics:

Main:

- IFCA
- FedCluster
- FedSC
- FedAC

Extra:

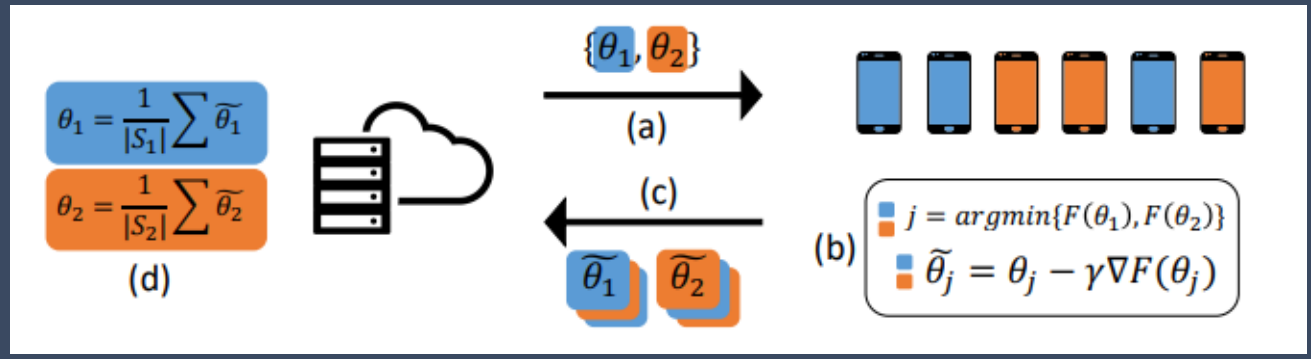
- FedACClient-centric FL (CC-FL)
- FedEff
- Fedsplit

IFCA

Idea:

1. Set k cluster centers
2. Find which cluster is closest
3. Update cluster parameters

Repeat the steps



IFCA

Formulation:

Let $f(\theta; z) : \Theta \to \mathbb{R}$ be the loss function associated with data point z, where $\Theta \subseteq \mathbb{R}^d$ is the parameter space. In this paper, we choose $\Theta = \mathbb{R}^d$. Our goal is to minimize the population loss function

$$F^j(\theta) := \mathbb{E}_{z \sim \mathcal{D}_j}[f(\theta; z)],$$

for all $j \in [k]$. For the purpose of theoretical analysis in Section 5, we focus on the strongly convex losses, in which case we can prove guarantees for estimating the unique solution that minimizes each population loss function. In particular, we try to find solutions $\{\widehat{\theta}_j\}_{j=1}^k$ that are close to

$$\theta_j^* = \text{argmin}_{\theta \in \Theta} F^j(\theta), \ j \in [k].$$

Since we only have access to finite data, we use empirical loss functions. In particular, let $Z \subseteq \{z^{i,1}, \ldots, z^{i,n}\}$ be a subset of the data points on the i-th machine. We define the empirical loss associated with Z as

$$F_i(\theta; Z) = \frac{1}{|Z|} \sum_{z \in Z} f(\theta; z).$$

When it is clear from the context, we may also use the shorthand notation $F_i(\theta)$ to denote an empirical loss associated with some (or all) data on the i-th worker.

Algorithm 1: One-shot clustering algorithm for federated learning

1: Worker machines send ERM $\widetilde{\theta}_i := \text{argmin}_{\theta \in \Theta} F_i(\theta)$ (for all $i \in [m]$) to the center.
2: Center machine clusters $\{\widetilde{\theta}_i\}_{i=1}^m$ into $S_1, \ldots, S_k$.
3: Within each cluster S_j, $j \in [k]$, run federated learning algorithm to obtain final solution $\widehat{\theta}_j$.

IFCA

Algorithm 2: Iterative Federated Clustering Algorithm (IFCA)

Input: number of clusters k, step size γ, $j \in [k]$, initialization $\theta_j^{(0)}$, $j \in [k]$
number of parallel iterations T, number of local gradient steps τ (for model averaging).
for $t = 0, 1, \ldots, T-1$ **do**
  center machine: broadcast $\theta_j^{(t)}$, $j \in [k]$
  $M_t \leftarrow$ random subset of worker machines (participating devices)
  for worker machine $i \in M_t$ in parallel **do**
    cluster identity estimate $\widehat{j} = \text{argmin}_{j \in [k]} F_i(\theta_j^{(t)})$
    define one-hot encoding vector $s_i = \{s_{i,j}\}_{j=1}^k$ with $s_{i,j} = \mathbf{1}\{j = \widehat{j}\}$
    option I (gradient averaging):
      compute (stochastic) gradient: $g_i = \widehat{\nabla} F_i(\theta_{\widehat{j}}^{(t)})$, send s_i, g_i to center machine
    option II (model averaging):
      $\widetilde{\theta}_i = \mathsf{LocalUpdate}(\theta_{\widehat{j}}^{(t)}, \gamma, \tau)$, send s_i, $\widetilde{\theta}_i$ to center machine
  end for
  center machine:
  option I (gradient averaging): $\theta_j^{(t+1)} = \theta_j^{(t)} - \frac{\gamma}{m} \sum_{i \in M_t} s_{i,j} g_i$, $\forall\, j \in [k]$
  option II (model averaging): $\theta_j^{(t+1)} = \sum_{i \in M_t} s_{i,j} \widetilde{\theta}_i / \sum_{i \in M_t} s_{i,j}$, $\forall\, j \in [k]$
end for
return $\theta_j^{(T)}$, $j \in [k]$

$\mathsf{LocalUpdate}(\widetilde{\theta}^{(0)}, \gamma, \tau)$ at the i-th worker machine
for $q = 0, \ldots, \tau - 1$ **do**
  (stochastic) gradient descent $\widetilde{\theta}^{(q+1)} = \widetilde{\theta}^{(q)} - \gamma \widehat{\nabla} F_i(\widetilde{\theta}^{(q)})$
end for
return $\widetilde{\theta}^{(\tau)}$

- The goal is to learn k cluster-specific models
- such that each client i is assigned to the model that best fits its local loss. Server updates each cluster model by aggregating updates only from clients assigned to that cluster

IFCA

Contributions

- Idea of clustering client data distributions
- Requires no shared data

Issues and future work

- choice of k
- Initialization of w_i
- communication cost

FedCluster

Idea: update model based on clusters

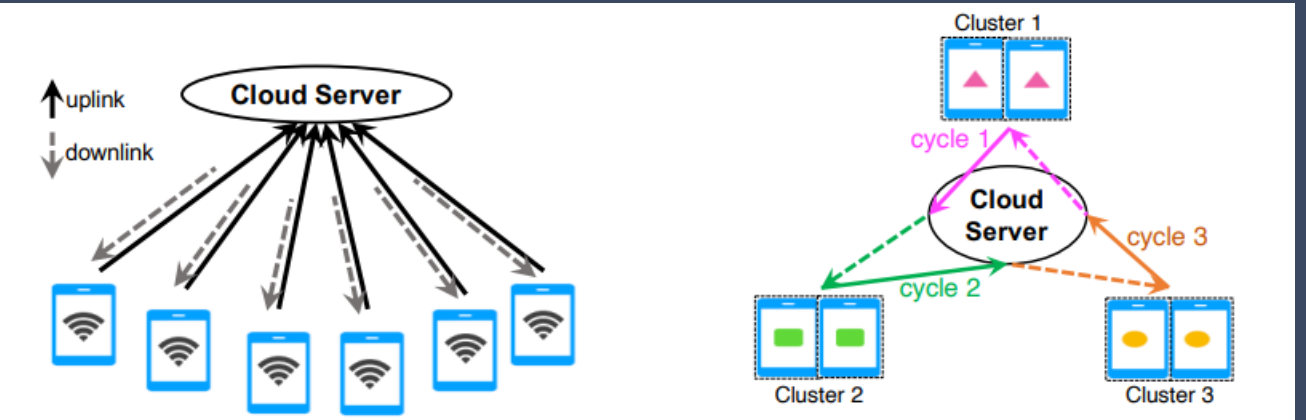


Fig. 1. Left: Traditional federated learning system. Right: FedCluster system with cluster-cycling.

FedCluster

Algorithm 1 FedCluster with local SGD

Input: Initialization model $W_0 \in \mathbb{R}^d$, learning rate $\eta_{j,K,t}$.
for rounds $j = 0, 1, \ldots, T-1$ **do**
  for cycles $K = 0, 1, ..., M-1$ **do**
    Sample a subset of devices $S_{K+1}^{(j)}$ from cluster $\mathcal{S}_{K+1}$.
    Cloud sends W_{jM+K} to the sampled devices.
    for all devices $k \in S_{K+1}^{(j)}$ **in parallel do**
      Initialize $w_{j,K,0}^{(k)} = W_{jM+K}$.
      for $t = 0, \ldots, E-1$ **do**
        Sample a local data point $\xi_{j,K,t}^{(k)} \in \mathcal{D}^{(k)}$ uniformly at random. Update $w_{j,K,t+1}^{(k)} = w_{j,K,t}^{(k)} - \eta_{j,K,t} \nabla f(w_{j,K,t}^{(k)}; \xi_{j,K,t}^{(k)})$.
      end
      Send the local model $w_{j,K,E}^{(k)}$ to the cloud.
    end
    Cloud computes $W_{jM+K+1} = \sum_{k \in \mathcal{S}'_{j,\sigma_j(K+1)}} \frac{p_k}{q_{\sigma_j(K+1)}} w_{j,K,E}^{(k)}$.
  end
end
Output: W_{TM}

FedCluster

Contribution:

Comparison: The key differences between the FedCluster framework and the traditional federated learning framework are in two-fold. First, in each learning round, FedCluster updates the global model multiple times (equals to number of clusters), whereas the traditional federated learning updates it only once. Hence, FedCluster is expected to make more optimization progress per learning round. In fact, as the devices in federated learning are usually busy and unavailable, the clusters in FedCluster provide much flexibility to schedule the learning tasks for the devices and make frequent updates on the global model (see the discussion in the next paragraph.). Second, the traditional federated learning process suffers from the device-level data heterogeneity. In comparison, as we show later in the analysis section, the convergence of the FedCluster learning process is affected by a smaller cluster-level data heterogeneity.

Issues and future work

- Number of clusters
- Determines clusters statically

FedSC

Idea: Use buttom-up clustering to group clients based on the similarity vectors from their model updates.

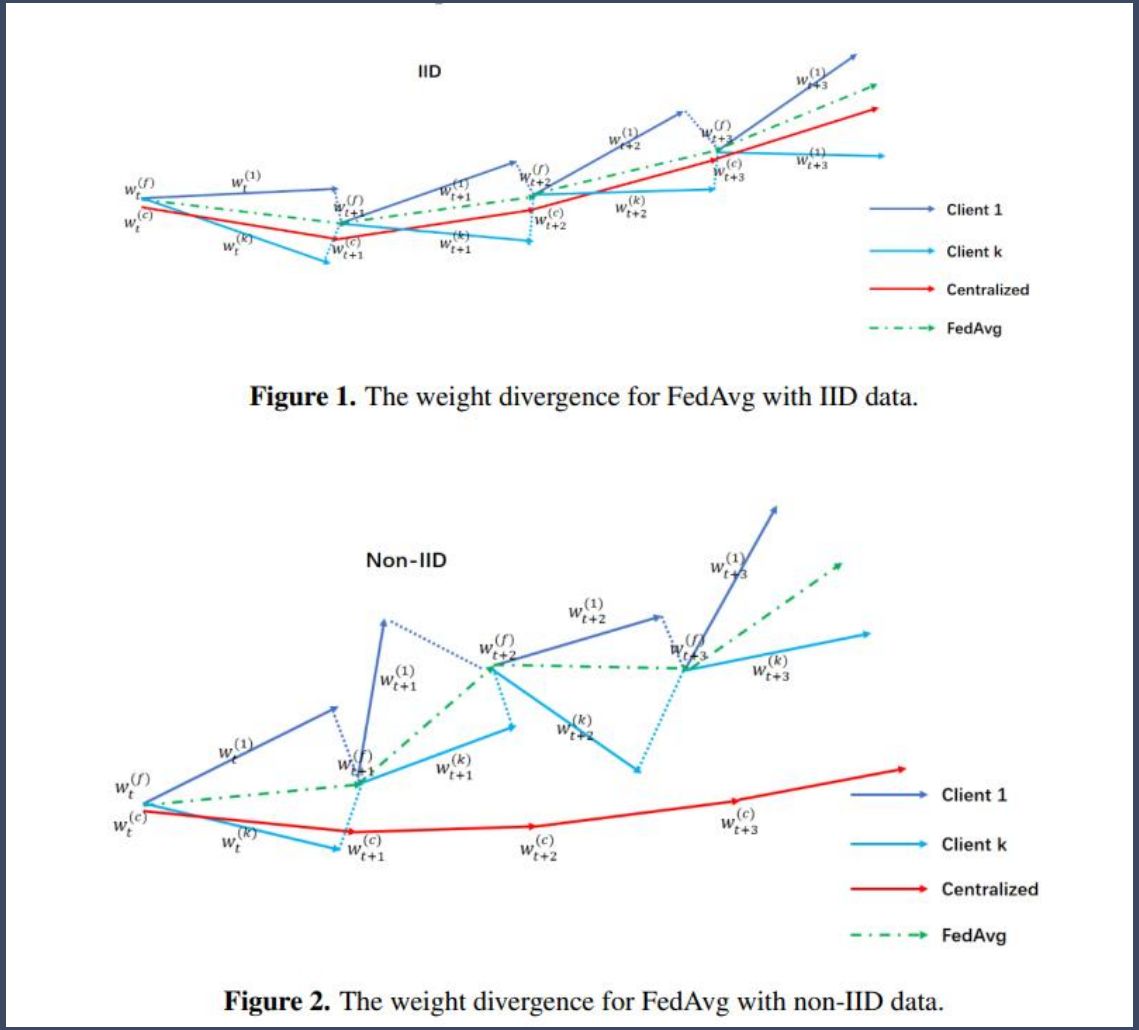


Figure 1. The weight divergence for FedAvg with IID data.

Figure 2. The weight divergence for FedAvg with non-IID data.

FedSC

3.3. Detailed steps are described below

First, each client sends its data attribute I to the central server. Because the information about the proportion of the client data categories may be leaked during this process, the client encrypts the data attribute I before sending it to the central server. After receiving data attributes from all clients and decrypting them, the central server uses the clustering algorithm to aggregate all clients into N clusters. Next, the central server selects a $C - fraction$ of clients from the first cluster and send the initial global model parameters $w_0^{(f)}$ to them. After the clients train E epochs with their private data, the obtained local model parameters are uploaded to the central server. The central server performs a weighted average of the obtained model parameters according to the number of data samples of the clients to obtain the global model parameter $w_1^{(f)}$. Then, the central server selects a $C - fraction$ of clients from the second cluster and sends the global model parameters to the selected clients. The selected client takes $w_1^{(f)}$ as the initial model parameter, uses the local dataset for training E epochs, and sends the trained model parameters to the central server. After obtaining the model parameters of all clients selected from the second cluster, the central server aggregates them to obtain the global model parameter $w_2^{(f)}$. Subsequently, $w_2^{(f)}$ is sent to the clients selected from the third cluster, and so on. This process is repeated for N aggregations, enabling the original model parameter to be trained with the data from N clusters and obtain the model parameter $w_N^{(f)}$. The model parameter $w_N^{(f)}$ contains the knowledge learned from N cluster data. This constitutes a communication process. Next, the central server sends the model parameters $w_N^{(f)}$ to the clients selected from the first cluster. This process is repeated until the model converges. In centralized machine learning, all data types are traversed in

3.1. Client clustering method based on local data type distribution

Non-IID data make local model parameters divergent in the federated learning so that the central server cannot aggregate a good model; however, federated learning can give satisfactory results with IID data. Thus, we cluster the clients according to their data distribution. The clients with high data distribution similarity are divided into a cluster; hence, data distribution in this cluster is similar to IID. At this time, federated optimization within each cluster can greatly improve the performance of the model and can improve the efficiency.

Let n_i be the total amount of data samples owned by client i, n_{ij} be the number of class j data samples owned by client i, and define the data attribute of client i as $I_i = \left[n_{i1}/n_i, n_{i2}/n_i, ..., n_{ij}/n_i\right]$. Clients are clustered based on their data attributes.

A bottom-up clustering algorithm is proposed. Initially, each client is treated as an individual cluster, and clusters are progressively merged at each step. Ultimately, a cluster encompassing all samples is obtained. The detailed algorithm steps are as follows:

1) Treat each client as an individual cluster.

2) Calculate the distance between two clusters and merge the cluster with the smallest distance. In this case, the distance between two clusters is defined as the maximum distance between any pair of clients in the two clusters. Specifically, the distance between cluster p and cluster q is defined as follows:

$$D_{p,q} = \max\left\{d_{ij} = \|I_i - I_j\|_2 \ | i \in p, j \in q\right\} \tag{3.1}$$

where d_{ij} represents the distance between the data attributes I_i of client i in cluster p and the data attributes I_j of client j in cluster q.

3) Repeat step 2 and iteratively aggregate clusters until the number of clusters meets the requirements.

In this way, the data distribution among clients within the same cluster becomes similar. Consequently, federated learning can effectively operate within each cluster, yielding improved results.

FedSC

Algorithm 1: FedSC algorithm

Input: Local dataset for each client
Output: FedSC algorithm
Each client sends its own data attributes I to the central server;
Server execution :
Cluster according to data attribute I of each client;
Initialize the global model parameter w_0;
for $t = 1, ..., T$ **do**
  Randomly select the participants of this round in each cluster S_t;
  $n \leftarrow \sum_{i \in S_t} |D_i|$;
  for $i \in S_t$ *in parallel* **do**
    Send the global model w_t to the selected participants;
    $\Delta w_i^t \leftarrow$ *client execution* (i, w_t);
  end
  $w_{t+1} \leftarrow w_t - \eta \sum_{i \in S_t} \frac{|D_i|}{n} \Delta w_k^t$;
end
return w_T
Client execution$(\mathbf{i}, \mathbf{w_t})$:
$w_i^t \leftarrow w_t$;
for *epoch* k=1,2,...,E **do**
  for *each batch* $b = \{x, y\}$ *of* D_i **do**
    $w_i^t \leftarrow w_i^t - \eta \nabla L\left(w_i^t; b\right)$;
  end
end
$\Delta w_i^t \leftarrow w_t - w_i^t$;
return Δw_i^t

- Builds a client similarity graph using cosine similarity of updates
- Bottom-up hierarchical clustering, Distance between clusters

FedSC

Contribution:

- Through experimental comparisons, the FedSC algorithm consistently achieves higher accuracy compared to alternative federated learning algorithms. Moreover, the FedSC algorithm ensures data security by completing the model training without sharing data among participating parties. This contribution further advances the development of federated learning.

Issues and future work

- Distance function
- Computation cost
- Clustering on server

FedSC: the more clusters, the better

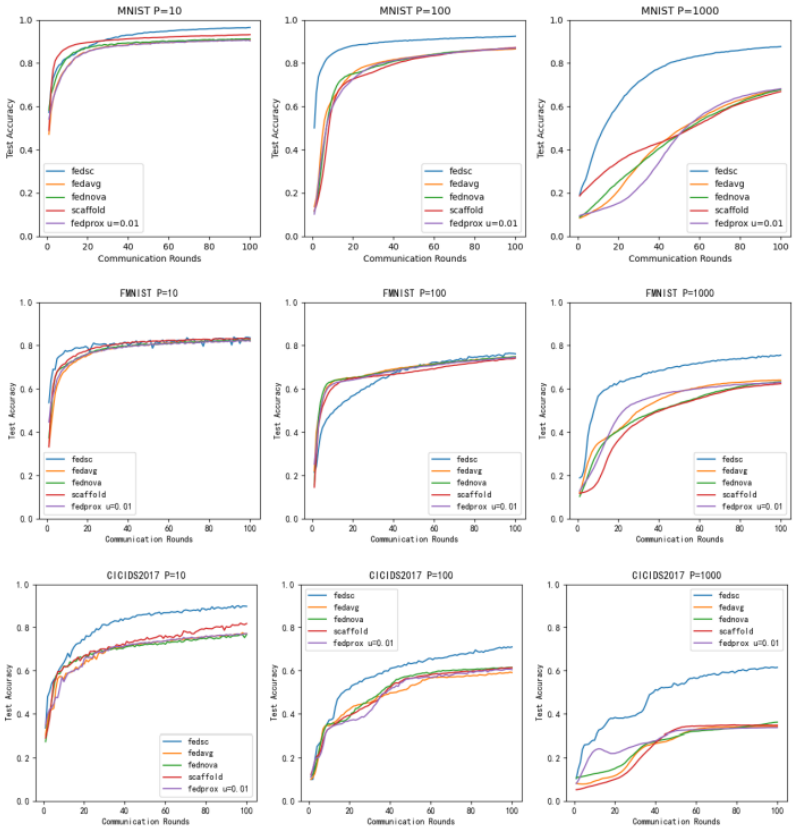


Figure 9. Training process of different approaches with different number of clients.

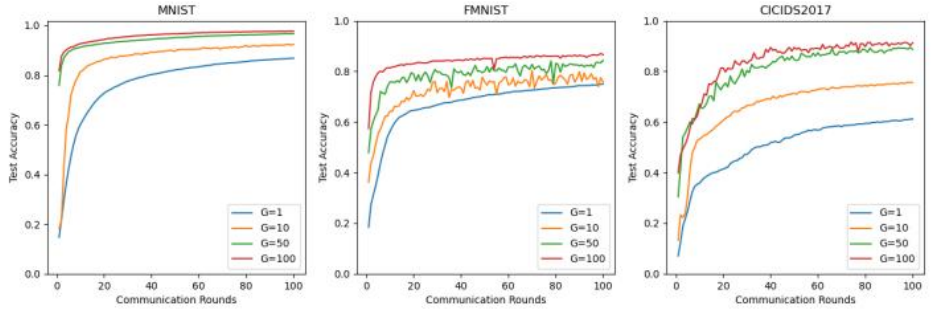


Figure 8. The influence of the number of cluster on the algorithm.

FedAC

Idea: Adaptive clustering

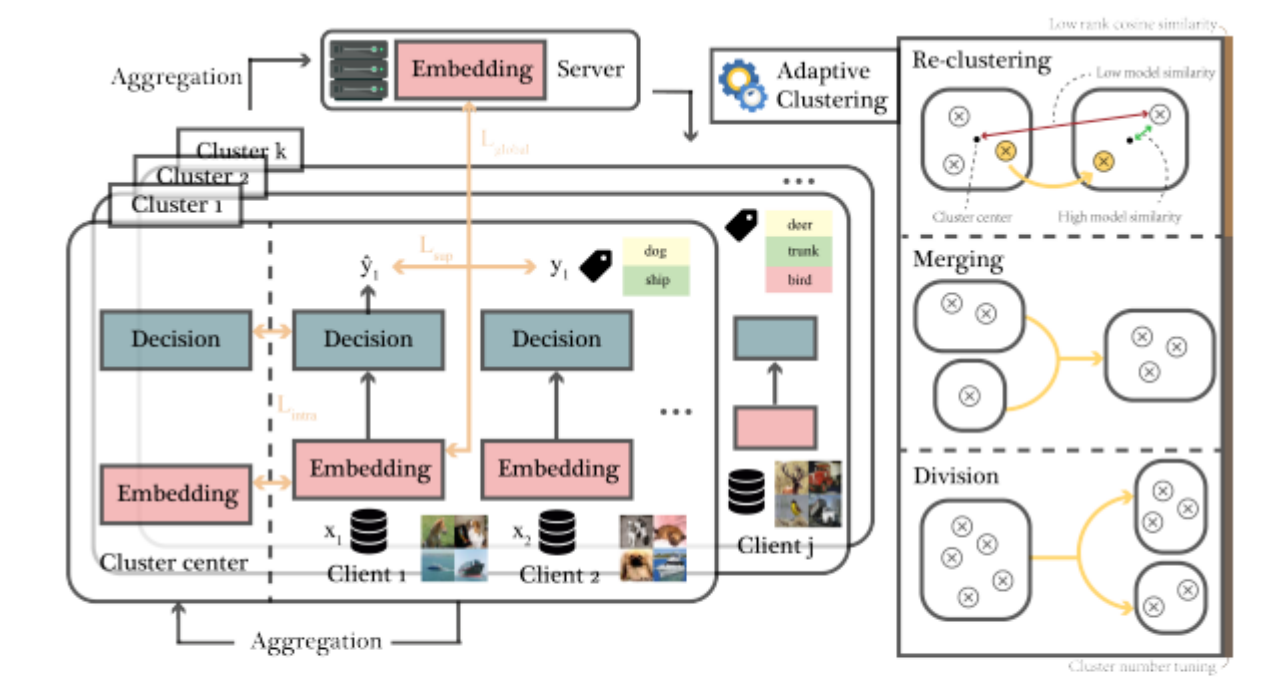


Fig. 1: The framwork of FedAC in the heterogeneous setting. The server clusters and aggregates clients to get a global embedding and multiple cluster center models. Clients update local models by minimizing the classification error loss (L_{sup}), intra-clustering regularization (L_{intra}), and global regularization (L_{global}).

FedAC

4.1 Overview and Optimization Objective

The proposed **Fed**erated Learning with **A**daptive **C**lustering (FedAC) addresses data heterogeneity and achieves efficient, adaptable client clustering. Firstly, to better integrate beneficial global knowledge into cluster, the model f is decoupled into two submodules, exemplified by convolutional neural networks (convnets): the embedding ϕ (comprising shallow layers with convolutional modules) and the decision h (comprising deep fully connected layers). Different aggregation strategies are designed for distinct submodules to represent global and intra-cluster knowledge, achieving optimal performance by balancing both aspects. Moreover, a cosine similarity metric based on dimensionality-reduced models is proposed for efficient online model similarity measurement to assist clustering throughout training. Additionally, a self-examining module evaluates the current cluster conditions and autonomously finetunes the cluster number. An overview of FedAC's framework is shown in Fig. 1.

FedAC

$$\min_{\{\omega_i\}} \frac{1}{m} \sum_{i=1}^{m} \{\mathcal{L}_i(\omega_i) + \frac{\mu}{2} \text{Regu}(\omega_i, \Omega_{k^*(i,R^*)})\},$$

$$\text{s.t.} \quad R^* \in \arg\max_{R} \frac{1}{m} \sum_{k=1}^{K} \sum_{i=1}^{m} R_{i,k} \text{Sim}(\omega_i, \Omega_k(\{\omega_i\}, R)),$$

In the context of FL, each client i possesses its private dataset $\mathcal{D}_i$ from distribution $\mathbb{P}_i(x, y)$, where x and y represent the input features and corresponding labels. In a vanilla setting (FedAvg), clients collectively contribute to a shared model $f(\omega; \cdot)$ parameterized by weights ω. The objective function is:

$$\min_{\omega} : \mathcal{F} = \sum_{i=1}^{m} \frac{|\mathcal{D}_i|}{N} \mathcal{L}_i(\omega), \tag{1}$$

where $\mathcal{L}_i(\omega) = \mathbb{E}_{(x,y)\sim\mathcal{D}_i} l(f(\omega; x); y)$ is the empirical loss of client i, m is the number of clients, and N denotes the total number of instances over all clients.

We apply L2 distance and the proposed low-rank cosine model similarity, *LrCos*, into Regu$(\cdot,\cdot)$ and Sim$(\cdot,\cdot)$, respectively, within Eq. 2, defining the optimization objective of FedAC:

$$\min_{\{\omega_i\}} \frac{1}{m} \sum_{i=1}^{m} \{\underbrace{\mathcal{L}_i(\omega_i)}_{L_{\text{sup}}} + \frac{\mu}{2} \underbrace{||\omega_i - \Omega_{k^*(i,R^*)}||_2^2}_{L_{\text{intra}}} + \frac{\lambda}{2} \underbrace{||\phi_i - \Phi(\{\phi_i\})||_2^2}_{L_{\text{global}}}\},$$

$$\text{s.t.} \quad R^* \in \arg\max_{R} \frac{1}{m} \sum_{k=1}^{K} \sum_{i=1}^{m} R_{i,k} LrCos(\omega_i, \Omega_k(\{\omega_i\}, R)), \tag{3}$$

4.4 Re-clustering

FedAC employs a EM-like algorithm for periodic clustering updates. In each E-step, clients are reassigned to nearby clusters based on the *LrCos* of their personalized models and cluster centroids:

$$R_{i,k} = \begin{cases} 1, & k = \arg\min_{j} LrCos(\omega_i, \Omega_j) \\ 0, & \text{else} \end{cases}. \tag{4}$$

In the M-step, the server aggregates the models within each cluster to obtain the cluster centroids:

$$\Omega_k = \frac{1}{\sum_{i=1}^{m} R_{i,k}} \sum_{i=1}^{m} R_{i,k} \omega_i. \tag{5}$$

FedAC

Algorithm 1 FedAC

Input: learning rate η, hyper-parameters μ and λ, initial number of clusters K
Output: $\{\omega_i\}_{i\in[m]}$
Server executes:

Initialize: R^0, $\{\omega_i^0 = (\phi_i^0, h_i^0)\}_{i\in[m]}$, $\{\Omega_k^0\}_{k\in[K]}$, Φ^0
for each round $t = 0, 1, \ldots$ **do**
  Randomly selects a subset of clients S_t
  for each client $i \in S_t$ **in parallel do**
    Server sends Ω_{k*}^t, Φ^t to client i
    $\omega_i^{t+1} \leftarrow$ LocalUpdate $(\omega_i^t, \Omega_{k*}^t, \Phi^t, \mu, \lambda, \eta)$
    Clients i sends ω_i^{t+1} back
  end for
  $\Phi^{t+1} = \sum_{i\in S_t} \phi_i^{t+1}$
  Calculate low-rank cosine model similarity {Algorithm 2}
  $R^{t+1} \leftarrow$ E-step $(\{\omega_i^{t+1}\}_{i\in[m]}, \{\Omega_k^t\}_{k\in[K]})$
  $\{\Omega_k^{t+1}\}_{k\in[K]} \leftarrow$ M-step $(\{\omega_i^{t+1}\}_{i\in[m]}, R^{t+1})$
  Cluster number tuning (CNT) {Algorithm 3}
end for

LocalUpdate $(\omega_i^t, \Omega_{k*}^t, \Phi^t, \mu, \lambda, \eta)$:

$\omega_i^{t,0} = \omega_i^t$
for each local epoch $r = 0,1,\ldots$ **do**
  Randomly selects a batch $\mathcal{B}_i$ from $\mathcal{D}_i$
  $\omega_i^{t,r+1} = \omega_i^{t,r} - \eta\nabla l_i(\omega_i^{t,r}; \mathcal{B}_i) - \eta\mu(\omega_i^{t,r} - \Omega_{k*}^t) - \eta\lambda(\phi_i^{t,r} - \Phi^t)$
end for
return $\omega_i^{t,r+1}$

Algorithm 2 Low-rank Cosine Similarity

Input: $\{\omega_i\}_{i\in[m]}$, $\{\Omega_k\}_{k\in[K]}$
Parameter: reduced number of dimensions D
Output: $LrCos_{i,j}$ between client i and cluster k

Mapping $M \leftarrow$ UpdateMap (S_t, D) {Proceed sparsely}
$LrCos_{i,j} = \frac{M\cdot\omega_i \cdot M\cdot\Omega_k}{||M\cdot\omega_i||_2||M\cdot\Omega_k||_2}$

UpdateMap (S_t, D):

$H \in \mathbb{R}^{dim(\omega)\times|S|} \leftarrow [\omega_1, \ldots, \omega_{|S|}]$
$M \in \mathbb{R}^{D\times dim(\omega)} \leftarrow$ PCA(H, components=D)
return M

Algorithm 3 Cluster Number Tuning (CNT)

Input: $\{\omega_i\}_{i\in[m]}$, $\{\Omega_k\}_{k\in[K]}$
Parameter: lower and upper threshold, a and b, of G_c

for each cluster $k \in [K]$ **do**
  $Dist_{intra}^k = \frac{1}{\sum_{i=1}^m R_{i,k}} \sum_{i=1}^m R_{i,k}||\omega_i - \Omega_k||_2^2$
  $Dist_{inter}^k = \frac{1}{K-1} \sum_{j=1}^K ||\Omega_k - \Omega_j||_2^2$
  $G_c^k = \frac{Dist_{intra}^k}{Dist_{inter}^k}$
  if $G_c^k < a$ **then**
    Merge cluster k into the closest cluster.
  else if $G_c^k > b$ **then**
    Divide cluster k into 2 clusters.
  end if
end for

References:

- Novel clustered federated learning based on local loss Endong Gu, Yongxin Chen, Hao Wen, Xingju Cai, Deren Han
- An Efficient Framework for Clustered Federated Learning Avishek Ghosh et all
- Multi-Center Federated Learning: Clients Clustering for Better Personalization Guodong Long, Ming Xie, Tao Shen, Tianyi Zhou, Xianzhi Wang and Jing Jiang1
- FedSC: A federated learning algorithm based on client-side clustering Zhuang Wang, Renting Liu*, Jie Xu and Yusheng Fu
- FedEff: efficient federated learning with optimal local epochs for heterogeneous clients K. Narmadha & P.Varalakshmi
- FedCluster: Boosting the Convergence of Federated Learning via Cluster-Cycling Cheng Chen et all
- FedAC: A Adaptive Clustered Federated Learning Framework for Heterogeneous Data Yuxin Zhang, Haoyu Chen, Zheng Lin, Zhe Chen, and Jin Zhao
- Client-Centric Federated Adaptive Optimization Jianhui Sun et all

Thanks

CC-FL

Idead: introduction of a new testbed

adaptive participation

personalized aggregation

We study general non-convex objective functions under statistical heterogeneity, i.e., each local loss $f_i(x)$ (and thus the global loss $f(x)$) may be non-convex [2], and $\mathcal{D}_i \neq \mathcal{D}_j$ when $i \neq j$.

Assumption 1 (Smoothness). Each local loss $f_i(x)$ has L-Lipschitz continuous gradients, i.e., $\forall x, x' \in \mathbb{R}^d$, we have $\|\nabla f_i(x) - \nabla f_i(x')\| \leq L\|x - x'\|$, where L is the Lipschitz constant. And f has finite optimal value, i.e., $f^* \triangleq \min_x f(x)$ exists, and $f^* > -\infty$.

Assumption 2 (Unbiased Bounded Gradient). We could access an unbiased estimator $g_{t,k}^i = \nabla f_i(x_{t,k}^i, \xi_{t,k}^i)$ of true gradient $\nabla f_i(x_{t,k}^i)$ for all t, k, i, where $g_{t,k}^i$ is the stochastic gradient with minibatch $\xi_{t,k}^i$. And the stochastic gradient is bounded, i.e., $\left\|g_{t,k}^i\right\| \leq G$.

Assumption 3 (Bounded Local Variance). Each stochastic gradient on the i-th client has a bounded local variance, i.e., we have $\mathbb{E}\left[\left\|g_{t,k}^i - \nabla f_i(x_{t,k}^i)\right\|^2\right] \leq \sigma_l^2$.

Assumption 4 (Bounded Global Variance). Local loss $\{f_i\}_{i=1}^n$ have bounded global variance, i.e., $\forall x$, $\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x) - \nabla f(x)\|^2 \leq \sigma_g^2$.

CC-FL

Algorithm 1: FedAvg [51], FedAvgM [27], FedAdam [58]

Input:
Number of clients n, objective function $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$;
Number of communication rounds T, **local** learning rate η_l, **local** updating number K;
Initialization x_0, **global** learning rate η, and β, γ, ϵ;

for $t \in \{1, ..., T\}$ **do**
  Randomly sample a subset $\mathcal{S}_t$ of clients;
  Server send x_t to subset $\mathcal{S}_t$ of clients;
  for *each client* $i \in \mathcal{S}_t$ **do**
    Initialize $x_{t,0}^i \leftarrow x_t$;
    for $k \in \{0, 1, ..., K-1\}$ **do**
      Randomly sample a batch $\xi_{t,k}^i$;
      Compute $g_{t,k}^i = \nabla f_i(x_{t,k}^i \ \xi_{t,k}^i)$;
      Update $x_{t,k+1}^i = x_{t,k}^i - \eta_l g_{t,k}^i$;
    end
    $\Delta_t^i = x_t - x_{t,K}^i$;
  end
  Server aggregates $\Delta_t = \frac{1}{|\mathcal{S}_t|}\sum_{i\in\mathcal{S}_t} \Delta_t^i$;
  Update $x_{t+1} = x_t - \eta\Delta_t$
  $m_t = (1-\beta)\Delta_t + \beta m_{t-1}$
  Update $x_{t+1} = x_t - \eta m_t$
  $m_t = (1-\beta)\Delta_t + \beta m_{t-1}$
  $v_t = (1-\gamma)\Delta_t^2 + \gamma v_{t-1}$
  Update $x_{t+1} = x_t - \eta \frac{m_t}{\sqrt{v_t}+\epsilon}$
end
return x_T

Algorithm 2: A class of **Client-Centric Federated Adaptive Optimization** approaches. CC-FedAdagrad CC-FedAdam CC-FedAMS

Input: Number of clients n, objective $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$, initialization x_0, Number of rounds T, **local** learning rate η_l, Number of participating clients m, server-side hyperparameter $\eta, \beta, \gamma, \epsilon$

for $t \in \{1, ..., T\}$ **do**
  At Each Client
    Self-determine whether to participate in the training, if not, stay idle.
    Once determine to participate, retrieve a global model x_μ from the server (μ may not be t)
    Trigger **Client-Centric Local Computation** (Algorithm 3), $\Delta_\mu^i =$ **CC-Local** (i, t, μ, η_l)
    Send local update Δ_μ^i
  At Server (Concurrently with Client)
    Collect the local updates $\{\Delta_{t-\tau_{t,i}}^i, i \in \mathcal{S}_t\}$ returned from a subset of clients $\mathcal{S}_t$, where $\tau_{t,i}$ represents the random delay of the client i's local update, $i \in \mathcal{S}_t$
    Aggregate: $\Delta_t = \frac{1}{m}\sum_{i\in\mathcal{S}_t} \Delta_{t-\tau_{t,i}}^i$
    Update momentum buffer $m_t = (1-\beta)\Delta_t + \beta m_{t-1}$;
    $\hat{v}_t = \hat{v}_{t-1} + \Delta_t^2$
    $\hat{v}_t = (1-\gamma)\Delta_t^2 + \gamma\hat{v}_{t-1}$
    $v_t = (1-\gamma)\Delta_t^2 + \gamma v_{t-1}$
    $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$
    Update $x_{t+1} = x_t - \eta \frac{m_t}{\sqrt{\hat{v}_t}+\epsilon}$;
end
return x_T

Algorithm 3: Client-Centric Local Computation CC-Local (i, t, μ, η_l)

Input: client index i, current round t, retrieved global model timestamp μ, **local** learning rate η_l

Initialize the local model $x_{\mu,0}^i = x_\mu$.
Determine a number of local steps $K_{t,i}$, which can be time-varying and device-dependent based on its own condition.
for $k \in \{0, 1, ..., K_{t,i} - 1\}$ **do**
  Randomly sample a batch $\xi_{\mu,k}^i$
  Compute $g_{\mu,k}^i = \nabla f_i(x_{\mu,k}^i; \xi_{\mu,k}^i)$
  Update $x_{\mu,k+1}^i = x_{\mu,k}^i - \eta_l g_{\mu,k}^i$
end
Compute model difference $\Delta_\mu^i = x_\mu - x_{\mu,K_{t,i}}^i$
Normalize w.r.t. $K_{t,i}$, and send $\Delta_\mu^i = \frac{\Delta_\mu^i}{K_{t,i}}$
return Δ_μ^i

CC-FL

Contribution:

- Claimed to present a novel and realistic idea

Issues and future work

- Does not provide any reason to declare superiority other than a more realistic approach
- Lacks strong theoretical justification for superiority

FedEff

Idea: variable number of epochs

$$E = \tau * E_b \quad where \quad \tau \in (0, 1]$$

$$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil$$

$$CT_i = ERT - (\beta_i + \delta_i)$$

$$E_i = \lfloor (CT_i / \sigma_i) \rfloor$$

Class	Epochs/Client	Epochs/Round	Description
EC	Same	Same	Epochs do not vary across clients and across rounds
UEC	Varies	Same	Epochs vary across clients but do not vary across rounds
EIC	Same	Varies	Epochs vary across rounds but do not vary across clients
UEIC	Varies	Varies	Epochs vary across clients and across rounds

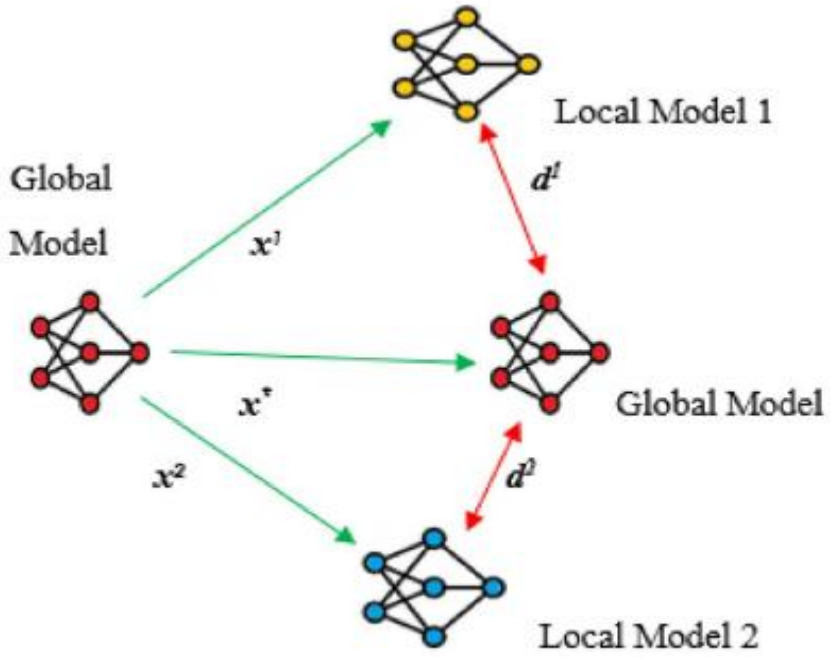


Fig. 1. Local(x^1,x^2,..) vs Global(x^*) parameters progression and divergence(d^1,d^2,..).

FedEff

To calculate GW_0, E_i; To estimate ERT
Initialization Round: Choose M, E_b**, b, R,** η
while each client i from 1 to M in parallel **do**
    $LW_i = Pretrainoneepoch(i, b)$
    $TT_i = \sigma_i + \beta_i + \delta_i$
    $\sigma_i = \frac{n_i}{b}(\frac{t}{CP_i} + \gamma_i)$
end while
$\sigma_a = \frac{1}{M}\sum_{i=1}^{M}\sigma_i$
$\beta_a = \frac{1}{M}\sum_{i=1}^{M}\beta_i$
$\delta_a = \frac{1}{M}\sum_{i=1}^{M}\delta_i$
$E = \tau * E_b$
$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil$
while each client i from 1 to M **do**
    $CT_i = ERT - (\beta_i + \delta_i)$
    $E_i = \lfloor (CT_i/\sigma_i) \rfloor$
end while
$GW_0 = \sum_{i=1}^{M}\frac{n_i}{n}LW_i$
Training Round:
while each round r from 1 to R **do**
    while each client i from 1 to M in parallel **do**
        $LW_i = ClientLocalUpdate(i, GW_{r-1}, E_i)$
    end while
    $GW_r = \sum_{i=1}^{M}\frac{n_i}{n}LW_i$
end while

Algorithm 2. (FedEff)

An Epoch Deciding Factor (EDF) τ, whose value ranges from (0,1], is optimally selected by the server based on the computation and communication speeds of all clients. With a suitable selection of the EDF, the ERT value can be varied accordingly. If the server wants a longer round time, it can choose higher values for τ closer to 1. Higher EDF values are meaningful in cases where the mean computation time is low and the difference between the maximum computation time (σ_{max}) and the minimum computation time (σ_{min}) is less. Lower EDF values are preferred if the server wants quick rounds. This value can be fixed or, if needed, can be adaptively chosen by the server, during training to decide the round time based on the availability of clients, stopping criteria of the federated training, run-time cost, etc. The EDF value is multiplied by the baseline epoch value (E_b) to find the approximate number of local epochs(E), which is used in the calculation of the ERT as shown in Equation (11).

$$E = \tau * E_b \quad where \quad \tau \in (0, 1] \tag{11}$$

The ERT is determined based on the assumption that the model is trained for E epochs, with the local training time σ_a and the two-way communication time $(\beta_a + \delta_a)$ as shown in Equation (12). A single training round consists of a download of global parameters, local model training, and an upload of updated local parameters. Thus, the approximate number of local epochs (E) can be fine-tuned based on the suitable selection of EDF.

$$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil \tag{12}$$

After calculating ERT, the server calculates the available computation time CT_i for each client i using Equation (13).

$$CT_i = ERT - (\beta_i + \delta_i) \tag{13}$$

Then, the number of optimal local epochs for all clients is computed using their respective computation times using Equation (14).

$$E_i = \lfloor (CT_i/\sigma_i) \rfloor \tag{14}$$

Contribution

- Optimized communication

drawbacks

- The paper claims “results in fast convergence of the global model” because“the total federated training time is reduced”

FedSplit

Idea: split operator, clients solve proximal subproblems; server performs averaging as a proximal operator

Based on Peaceman-Rachford Splitting

1.1 Notation

For the reader's convenience, we collect here our notational conventions.

Set and vector arithmetic: Given vectors $x, y \in \mathbf{R}^d$, we use $x^\mathsf{T} y = \sum_{j=1}^d x_j y_j$ to denote their Euclidean inner product, and $\|x\| = \sqrt{x^\mathsf{T} x}$ to denote the Euclidean norm. Given two non-empty subsets $A, B \subset \mathbf{R}^d$, their Minkowski sum is given by $A + B = \{x + y \mid x \in A, y \in B\}$. We also set $x + B = \{x\} + B$ for any point $x \in \mathbf{R}^d$.

For an integer $m \geqslant 1$, we use the shorthand $[m] := \{1, \dots, m\}$. Given a block-partitioned vector $z = (z_1, \dots, z_m) \in (\mathbf{R}^d)^m$ with $z_j \in \mathbf{R}^d$ for $j \in [m]$, we define the block averaged vector $\overline{z} := \frac{1}{m} \sum_{j=1}^m z_j$. Very occasionally, we also slightly abuse notation by defining arithmetic between vectors of dimension with a common factor. For example, if $x \in \mathbf{R}^d$ and $(y_1, \dots, y_m) = y \in (\mathbf{R}^d)^m$, then

$$x + y := (x + y_1, \dots, x + y_m).$$

Regularity conditions: A differentiable function $f : \mathbf{R}^d \to \mathbf{R}$ is said to be ℓ-strongly convex if

$$f(y) \geqslant f(x) + \nabla f(x)^\mathsf{T}(y - x) + \frac{\ell}{2}\|y - x\|^2, \quad \text{for all} \quad x, y \in \mathbf{R}^d.$$

It is simply convex if this condition holds with $\ell = 0$. Similarly, a differentiable function $f : \mathbf{R}^d \to \mathbf{R}$ is L-smooth if its gradient ∇f is L-Lipschitz continuous,

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|, \quad \text{for all } x, y \in \mathbf{R}^d.$$

Operator notation: Given an operator $\mathcal{T} : \mathbf{R}^d \to \mathbf{R}^d$ and a positive integer k, we use $\mathcal{T}^k$ to denote the composition of $\mathcal{T}$ with itself k times—that is, $\mathcal{T}^k$ is a new operator that acts on a given $x \in \mathbf{R}^d$ as $\mathcal{T}^k x := \underbrace{\mathcal{T} \circ \mathcal{T} \circ \cdots \circ \mathcal{T}}_{k \text{ times}} x$. An operator $\mathcal{T} : \mathbf{R}^d \to \mathbf{R}^d$ is said to be *monotone* if

$$(\mathcal{T}y - \mathcal{T}x)^\mathsf{T}(y - x) \geqslant 0 \qquad \text{for all } x, y \in \mathbf{R}^d. \tag{2}$$

FedSplit

3 A splitting framework and convergence guarantees

We now turn to the description of a framework that allows us to provide a clean characterization of the fixed points of iterative algorithms and to propose algorithms with convergence guarantees. Throughout our development, we assume that each function $f_j : \mathbf{R}^d \to \mathbf{R}$ is convex and differentiable.

3.1 An operator-theoretic view

We begin by recalling the consensus formulation (3) of the problem in terms of a block-partitioned vector $x = (x_1, \ldots, x_m) \in (\mathbf{R}^d)^m$, the function $F : (\mathbf{R}^d)^m \to \mathbf{R}$ given by $F(x) := \sum_{j=1}^m f_j(x_j)$, and the constraint set $E := \{x \mid x_1 = x_2 = \cdots = x_m\}$ is the feasible subspace for problem (3). By appealing to the first-order optimality conditions for the problem (3), it is equivalent to find a vector $x \in (\mathbf{R}^d)^m$ such that $\nabla F(x)$ belongs to the normal cone of the constraint set E, or equivalently such that $\nabla F(x) \in E^\perp$. Equivalently, if we define a set-valued operator $\mathcal{N}_E$ as

$$\mathcal{N}_E(x) := \begin{cases} E^\perp, & x_1 = x_2 = \cdots = x_m, \\ \emptyset, & \text{else} \end{cases} \tag{14}$$

then it is equivalent to find a vector $x \in (\mathbf{R}^d)^m$ that satisfies the inclusion condition

$$0 \in \nabla F(x) + \mathcal{N}_E(x). \tag{15}$$

where $\nabla F(x) = (\nabla f_1(x_1), \ldots, \nabla f_m(x_m))$.

When the loss functions $f_j : \mathbf{R}^d \to \mathbf{R}$ are convex, both ∇F and $\mathcal{N}_E$ are monotone operators on $(\mathbf{R}^d)^m$, as defined in equation (2). Thus, the display (15) is a *monotone inclusion problem*. Methods for solving monotone inclusions have a long history of study within the applied mathematics and optimization literatures [28, 7]. We now use this framework to develop and analyze algorithms for solving the federated problems of interest.

In understanding these and other algorithms, it is convenient to introduce the consensus reformulation of the distributed problem (1), which takes the form

$$\begin{array}{ll} \text{minimize} & F(x) := \sum_{j=1}^m f_j(x_j) \\ \text{subject to} & x_1 = x_2 = \cdots = x_m. \end{array} \tag{3}$$

Although this consensus formulation involves more variables, it is more amenable to the analysis of distributed procedures [5].

Algorithm 1 [FedSplit] *Splitting scheme for solving federated problems of the form* (1)

Given initialization $x \in \mathbf{R}^d$, proximal solvers $\texttt{prox\_update}_j : \mathbf{R}^d \to \mathbf{R}^d$
Initialize $x^{(1)} = z_1^{(1)} = \cdots = z_m^{(1)} = x$
for $t = 1, 2, \ldots$:
1. **for** $j = 1, \ldots, m$:
   a. *Local prox step:* set $z_j^{(t+1/2)} = \texttt{prox\_update}_j(2x^{(t)} - z_j^{(t)})$
   b. *Local centering step:* set $z_j^{(t+1)} = z_j^{(t)} + 2(z_j^{(t+1/2)} - x^{(t)})$
   end for
2. *Compute global average:* set $x^{(t+1)} = \overline{z}^{(t+1)}$.

end for

FedSplit

Contribution:

Comparison: The key differences between the FedCluster framework and the traditional federated learning framework are in two-fold. First, in each learning round, FedCluster updates the global model multiple times (equals to number of clusters), whereas the traditional federated learning updates it only once. Hence, FedCluster is expected to make more optimization progress per learning round. In fact, as the devices in federated learning are usually busy and unavailable, the clusters in FedCluster provide much flexibility to schedule the learning tasks for the devices and make frequent updates on the global model (see the discussion in the next paragraph.). Second, the traditional federated learning process suffers from the device-level data heterogeneity. In comparison, as we show later in the analysis section, the convergence of the FedCluster learning process is affected by a smaller cluster-level data heterogeneity.

Issues and future work

This paper leaves open a number of questions. First of all, the analysis of this paper was limited to deterministic algorithms, whereas in practice, one typically uses stochastic approximations to gradient updates. In the context of the `FedSplit` procedure, it is natural to consider stochastic approximations to the proximal updates that underlie it. Given our results on the incorrectness of previously proposed methods and the work of Woodworth and colleagues [30] on the suboptimality on multi-step stochastic gradient methods, it is interesting to develop a precise characterization of the tradeoff between the accuracy of stochastic and deterministic approximations to intermediate quantities and rates of convergence in federated optimization. It is also interesting to consider stochastic approximation methods that exploit higher-order information, such as the Newton sketch algorithm and other second-order subsampling procedures [22, 23].