Towards Understanding the Influence of Individual Clients in Federated Learning

Yihao Xue [1], Chaoyue Niu [1], Zhenzhe Zheng [1],
Shaojie Tang [2], Chengfei Lyu [3], Fan Wu [1], and Guihai Chen [1]

[1] Shanghai Jiao Tong University,
[2] The University of Texas at Dallas,
[3] Alibaba Group
yh_xue@outlook.com, rvince@sjtu.edu.cn, zhengzhenzhe@sjtu.edu.cn,
shaojie.tang@utdallas.edu, chengfei.lcf@alibaba-inc.com, fwu@cs.sjtu.edu.cn, gchen@cs.sjtu.edu.cn

Abstract

Federated learning allows mobile clients to jointly train a global model without sending their private data to a central server. Extensive works have studied the performance guarantee of the global model, however, it is still unclear how each individual client influences the collaborative training process. In this work, we defined a new notion, called *Fed-Influence*, to quantify this influence over the model parameters, and proposed an effective and efficient algorithm to estimate this metric. In particular, our design satisfies several desirable properties: (1) it requires neither retraining nor retracing, adding only linear computational overhead to clients and the server; (2) it strictly maintains the tenets of federated learning, without revealing any client's local private data; and (3) it works well on both convex and non-convex loss functions, and does not require the final model to be optimal. Empirical results on a synthetic dataset and the FEMNIST dataset demonstrate that our estimation method can approximate Fed-Influence with small bias. Further, we show an application of Fed-Influence in model debugging.

Introduction

Federated learning ingeniously leverages a large amount of valuable data in a distributed manner, while mitigating systemic privacy risks (McMahan et al. 2017; Kairouz et al. 2019). In such a setting, the training data are stored on multiple decentralized clients, who share a common global model and train it collaboratively. There is a central server that orchestrates the whole training process. In each round, the server collects local models from some eligible clients and use them to update the global model.

In this paper, we consider a significantly important but less considered problem in federated learning: *how does each client influence the global model?* Finding the answer to this question is meaningful in several aspects. On one hand, it helps us understand where the model comes from by explaining the prediction of the current global model in terms of clients. On the other hand, it provides quantitative insights into the roles of individual clients in federated learning, and informs us whether the existence of a certain client benefits the global model. This is critical to a fair credit/reward allocation, and, more importantly, debugging for federated learning. Thus, a fine-grained understanding of clients' influence further facilitates the exclusion of negative-influence clients or dynamically requires them to check their local data, thereby improving model performance. All of these are important to the explanation and robustness of federated learning, and also help sustain long-term user participation.

There already exists a classical statistics notion of "influence" in centralized learning, which evaluates the effect that the absence of an individual sample has on a model. To measure this influence, one should conduct a leave-one-out test (Cook 1977): retrain the model over the training set with one certain sample removed, and compare this model with that trained on the full dataset. A notion from robust statistics, called "influence function" (Jaeckel 1972; Hampel 1974; Cook and Weisberg 1980), was introduced to avoid retraining by measuring the change in the model caused by slightly changing the weight of one sample and using quadratic approximation combined with a Newton step (Cook and Weisberg 1982). Koh and Liang (2017) leveraged influence function in modern machine learning settings and developed an efficient and simple implementation using second-order optimization techniques. Hara, Nitanda, and Maehara (2019) went beyond convexity and optimality, which are two important assumptions in (Koh and Liang 2017). Koh et al. (2019) studied the effects of removing a group of data points, which is analogous to removing a client that holds a subset of training data in federated learning, except that their study is still in a centralized setting. Khanna et al. (2019) applied Fisher kernels along with sequential Bayesian quadrature to identify a subset of training examples that are most responsible for a given set of predictions and recovered (Koh and Liang 2017) as a special case.

The existing works above focused on centralized learning, and we are the first to consider a similar problem, the influence of individual clients, under a brand new framework, namely federated learning. There are some essential differences between centralized learning and federated learning which raise several design challenges: (1) The server in centralized learning, as the influence evaluator, has the full control over the considered sampling data, while in federated learning, the server would not be able to access clients' raw

data because of the privacy requirement; (2) the clients in federated learning may not always be available, due to the unreliable network connection. This implies that the server cannot communicate with a certain client at any desired time; and (3) the computing resources of mobile clients are limited, for which reason we should not bring too many additional computational burdens to clients when measuring their influence.

Due to the first difference, sample-level influence in centralized learning cannot be applied to measuring the influence of individual clients in federated learning. In this work, we turn to client-level influence measurement by investigating the effect of removing a client. In addition, works in centralized learning mainly focus on the influence on a model's testing loss, while we consider the influence on the parameter of the global model for the following two reasons: (1) In centralized learning, to cut down the time complexity, some techniques can be applied to obtain influence on loss without computing that on parameter (Hara, Nitanda, and Maehara 2019). However, in federated learning, computing influence on parameter is unavoidable because of the second and third differences mentioned earlier. Please refer to the supplement [1] for detailed reasoning; and (2) influence on parameter is more fundamental and powerful than that on loss. With the knowledge of influence on parameter, we can easily derive the influence of individual clients in terms of various metrics for evaluating models, such as loss, accuracy, precision, etc.

Our contributions. (1) To the best of our knowledge, we are the first to consider client-level influence in federated learning; (2) we propose a basic estimator for individual clients' influence on model parameter by leveraging the relationship between the global models in two consecutive communication rounds. Guided by the error analysis, we extend the basic design to support both convex and non-convex loss functions. We also develop an efficient implementation, bringing only slight communication and computation overhead to the server and the clients (in fact no extra computation overhead for the client); and (3) empirical studies on a synthetic dataset and the FEMNIST dataset (Caldas et al. 2018) demonstrate the effectiveness of our method. The estimation error observed in experiments indicates both the necessity and accuracy of our method. Based on the influence on parameter, we further derive the influence on model performance and observe that it was well estimated. In particular, the Pearson correlation coefficient between the estimated influence on loss and the ground truth achieves 0.62 in the most difficult setting. We also leverage influence on model performance for client valuation and client cleansing.

Problem Formulation

Federated Learning

We first introduce some necessary notations. Let $\mathcal{C}$ denote the set of all the clients, and $\mathcal{D}^k$ denote the local dataset of client $k \in \mathcal{C}$ with n_k samples. The set $\mathcal{D} = \bigcup_{k\in\mathcal{C}} \mathcal{D}^k$ is the full training set. For any set of clients $\mathcal{C}'$, we use $N(\mathcal{C}') = \sum_{k\in\mathcal{C}'} n_k$ to denote the total size of these clients' datasets. $\mathcal{L}(\mathbf{w}, z)$ denotes the loss function over a model $\mathbf{w}$ and a sample z. In addition, $\mathcal{L}(\mathbf{w}, \mathcal{D}^k) = \frac{1}{n_k}\sum_{z\in\mathcal{D}^k}\mathcal{L}(\mathbf{w}, z)$ denotes the empirical loss over a model $\mathbf{w}$ and a dataset $\mathcal{D}^k$. Then, we consider the following optimization task of federated learning:

$$\min_{\mathbf{w}\in\mathbb{R}^{\mathbf{p}}} \left\{ \mathcal{L}(\mathbf{w}, \mathcal{D}) = \sum_{k\in\mathcal{C}} \frac{n_k}{N(\mathcal{C})} \mathcal{L}(\mathbf{w}, \mathcal{D}^k) \right\}, \tag{1}$$

where the global loss function $\mathcal{L}(\mathbf{w}, \mathcal{D})$ is the weighted average of the local functions $\mathcal{L}(\mathbf{w}, \mathcal{D}^k)$ with the weight proportioning to the size of each client's local dataset. In this work, we consider a standard algorithm, federated averaging (FedAvg) (McMahan et al. 2017), to solve the optimization problem in (1). Although there are some other variants, such as FedBoost (Hamer, Mohri, and Suresh 2020), FedNova (Wang et al. 2020), FetchSGD (Rothchild et al. 2020), FedProx (Li et al. 2020a), and SCAFFOLD (Karimireddy et al. 2019), FedAvg is the first and the most widely used one. As a result, we see FedAvg as our basic block, which executes as follows. In the initial stage, the server randomly initializes a global model $\mathbf{w}_0$. Then, the training process is orchestrated by repeating the following two steps within each communication round t from 1 to T:

- **Local training.** The server selects a random set of clients $\mathcal{C}_t$ as participants in this round. Each participant $k \in \mathcal{C}_t$ then downloads from the server the latest global model $\mathbf{w}_{t-1}$, *i.e.*, the output model from the previous round $t-1$. Then, the client k performs local updates for each local iteration i from 1 to m:
$$\mathbf{w}_{t,i}^k \leftarrow \mathbf{w}_{t,i-1}^k - \eta \nabla_w \mathcal{L}\left(\mathbf{w}_{t,i-1}^k, \mathcal{D}^k\right), \tag{2}$$
with the starting local model $\mathbf{w}_{t,0}^k$ initialized as $\mathbf{w}_{t-1}$. In addition, η denotes the learning rate.
- **Model aggregation.** Participants in round t upload their updated local models. The server aggregates (takes a weighted average of) the local models to a new global model $\mathbf{w}_t$:
$$\mathbf{w}_t \leftarrow \sum_{k\in\mathcal{C}_t} \frac{n_k}{N(\mathcal{C}_t)} \mathbf{w}_{t,m}^k, \tag{3}$$
where the weight of client k is the size of her dataset n_k.

Fed-Influence

To express the client-level influence on federated learning clearly, we introduce a new notation $\mathbf{w}_t\left(\mathcal{C}' \to \mathcal{C}'\backslash\{c\}\right)$, which represents the aggregated model in round t with a client set $\mathcal{C}'$ replaced with $\mathcal{C}'\backslash\{c\}$, where $\mathcal{C}' \in \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_T, \mathcal{C}\}$. For example, $\mathbf{w}_{10}\left(\mathcal{C}_5 \to \mathcal{C}_5\backslash\{c\}\right)$ represents the resulting model we get at the end of round 10 if we remove the client c in round 5. One special case is that $\mathcal{C}' = \mathcal{C}_t$, *i.e.*,

$$\mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t\backslash\{c\}\right) = \sum_{k\in\mathcal{C}_t\backslash\{c\}} \frac{n_k}{N\left(\mathcal{C}_t\backslash\{c\}\right)} \mathbf{w}_{t,m}^k. \tag{4}$$

Another special case is $\mathbf{w}_t\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right)$, where we permanently remove the client c, *i.e.*, replace $\mathcal{C}_t$ with $\mathcal{C}_t\backslash\{c\}$ for all the rounds $t \in [T]$.

[1] The supplement is available from https://drive.google.com/file/d/1zFefHCAYiv5DPJ6nDVjgLUeO4yOoAFlV/view?usp=sharing.

With these notations, we next give the following definition of *Fed-Influence on Parameter (FIP)*.

Definition 1. *We refer to the change in model parameters due to removing a client c from $\mathcal{C}$ as Fed-Influence on Parameter (FIP) of client c, denoted by $\epsilon_t^{-c,*}$:*

$$\epsilon_t^{-c,*} \stackrel{\text{def}}{=} \mathbf{w}_t\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_t. \tag{5}$$

Based on FIP, we can extend the notion to measure the influence on model performance, such as the metrics of accuracy, cross-entropy loss, precision, recall, mean squared error (MSE), and etc. We can derive the influence on any of these metrics from FIP. Suppose $\mathcal{F}$ is the loss function for a certain metric over a test set $\mathcal{D}_{test}$, we can compute

$$\mathcal{F}\left(\mathbf{w}_t + \epsilon_t^{-c,*}, \mathcal{D}_{test}\right) - \mathcal{F}\left(\mathbf{w}_t, \mathcal{D}_{test}\right) \tag{6}$$

as the Fed-Influence over the metric. In Sections and , we will focus on two most widely used metrics, loss and accuracy, which corresponds to Fed-Influence on Loss (FIL) and Fed-Influence on Accuracy (FIA), respectively.

The exact value of FIP for a client can only be obtained by conducting leave-one-out test: retrain the model by removing the client, and compare the retrained model with the model trained on the full client set. However, it is prohibitively inefficient to rerun the whole federated learning process, especially when we intend to measure the influence of each client, implying the number of rerunning being the total number of clients.

Basic Estimator

We now derive an estimator for FIP $\epsilon_t^{-c,*}$ to avoid retraining. We start by rewriting the expression of $\epsilon_t^{-c,*}$ as follows

$$\begin{aligned}
\epsilon_t^{-c,*} &\stackrel{\text{def}}{=} \mathbf{w}_t\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_t \\
&= \mathbf{w}_t\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t\backslash\{c\}\right) \\
&\quad + \mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t\backslash\{c\}\right) - \mathbf{w}_t \\
&= \underbrace{\sum_{k\in\mathcal{C}_t\backslash\{c\}} \frac{n_k}{N(\mathcal{C}_t\backslash\{c\})} \underbrace{\left(\mathbf{w}_{t,m}^k\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_{t,m}^k\right)}_{\text{local sequential influence}}}_{\text{sequential influence}} \\
&\quad + \underbrace{\mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t\backslash\{c\}\right) - \mathbf{w}_t}_{\text{combinatorial influence}}
\end{aligned} \tag{7}$$

Equation (7) shows that $\epsilon_t^{-c,*}$ comprises three parts: (1) *Local sequential influence*: the influence that removing client c from the client set $\mathcal{C}$ on the local model of any other participating client $k \in \mathcal{C}_t\backslash\{c\}$ in round t. We regard it as "sequential" influence because it can be derived from $\epsilon_{t-1}^{-c,*}$, the influence in the previous round; (2) *Sequential influence*: the weighted average of local sequential influence; and (3) *Combinatorial influence*: the influence of removing c merely from $\mathcal{C}_t$ in the round t. The combinatorial influence is "combinatorial" as it is independent of $\epsilon_{t-1}^{-c,*}$.

We next dissect how to compute local sequential influence and combinatorial influence. The combinatorial one can be easily obtained using (3) and (4). The local sequential influence is more complicated. Given that the reasons behind the difference between $\mathbf{w}_{t,m}^k\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right)$ and $\mathbf{w}_{t,m}^k$ is that they are locally updated from different initial models, $\mathbf{w}_{t-1}\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right)$ and $\mathbf{w}_{t-1}$, respectively. We take a look at the first local iteration at round t. we estimate the term by applying first-order Taylor approximation and the chain rule:

$$\begin{aligned}
&\mathbf{w}_{t,m}^k\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_{t,m}^k \\
&\approx \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,m-1}^k} \frac{\partial \mathbf{w}_{t,m-1}^k}{\partial \mathbf{w}_{t,m-2}^k} \cdots \frac{\partial \mathbf{w}_{t,1}^k}{\partial \mathbf{w}_{t,0}^k} \Delta \mathbf{w}_{t,0}^k
\end{aligned} \tag{8}$$

where $\Delta \mathbf{w}_{t,0}^k = \mathbf{w}_{t,0}^k\left(\mathcal{C} \to \mathcal{C}\backslash\{c\}\right) - \mathbf{w}_{t,0}^k = \epsilon_{t-1}^{-c,*}$. According to the update rule in Equation 2, with the assumption that $\mathcal{L}(\mathbf{w}, \mathcal{D}^k)$ is twice differentiable, we obtain

$$\frac{\partial \mathbf{w}_{t,i}^k}{\partial \mathbf{w}_{t,i-1}^k} = \mathbf{I} - \eta \mathbf{H}_{t,i-1}^k, \tag{9}$$

where $\mathbf{H}_{t,i}^k \stackrel{\text{def}}{=} \nabla_w^2 \mathcal{L}(\mathbf{w}_{t,i}^k, \mathcal{D}^k)$. By combining Equations 7, 8 and 9, we get an estimator of $\epsilon_t^{-c,*}$, denoted by ϵ_t^{-c}

$$\epsilon_t^{-c} \stackrel{\text{def}}{=} \mathbf{M}_t^{-c} \epsilon_{t-1}^{-c} + \mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t\backslash\{c\}\right) - \mathbf{w}_t, \tag{10}$$

where

$$\mathbf{M}_t^{-c} \stackrel{\text{def}}{=} \sum_{k\in\mathcal{C}_t\backslash\{c\}} \frac{n_k}{N(\mathcal{C}_t\backslash\{c\})} \prod_{i=0}^{m-1} \left(\mathbf{I} - \eta \mathbf{H}_{t,i}^k\right). \tag{11}$$

By recalling that the initial model $\mathbf{w}_0$ is randomly initialized by the server, we have $\epsilon_0^{-c} = \mathbf{0}$. Then the estimator ϵ_t^{-c} can be computed iteratively using Equation 10.

We finally take a close look at the relation between the estimation error and the iteration t. We give a uniform bound on the error in both convex and non-convex cases under the following assumptions.

Assumption 1. *There exists λ and Λ such that $\lambda I \preccurlyeq \nabla^2 \mathcal{L} \preccurlyeq \Lambda I$.*

Assumption 2. *The norm of $\epsilon_t^{-c,*}$ is bounded by C, for $t \in [T]$ and $c \in \mathcal{C}$.*

Note that in Assumption 1, there is no constraint on the values of λ and Λ except $\lambda \leq \Lambda$, which means the loss function is not necessarily convex. Next we give Theorem 1, the proof of which is provided in the supplement.

Theorem 1. *With Assumptions 1 and 2, the error of the estimator is bounded by*

$$\delta_t^{-c} \stackrel{\text{def}}{=} \|\epsilon_t^{-c,*} - \epsilon_t^{-c}\| \leq \frac{1-\gamma^t}{1-\gamma} o(C), \quad \forall t > 0, \tag{12}$$

where o is the little-o notation, and $\gamma = \alpha^m, \alpha = \max\{|1 - \eta\lambda|, |1 - \eta\Lambda|\}$.

From (12), we can observe that the bound is in the format of the sum of geometric series. An intuitive explanation is that each time we use (10), the error in the previous round is scaled by $\mathbf{M}_t^{-c}$, and added a newly introduced error in this round, similar to summing a geometric series. Finally, there are three different cases depending on the relation between γ and 1:

- **Case 1** ($\gamma < 1$): In this case, $\lambda > 0$ and $0 < \eta < \frac{2}{\Lambda}$, where the loss function is strongly-convex and the learning rate is small enough. This is the most ideal case, where the bound can be further scaled to $\frac{1}{1-\gamma}o(C)$, which is independent of t.
- **Case 2** ($\gamma = 1$): In this case, either $\lambda = 0$ and $\eta \leq \frac{2}{\Lambda}$ or $\lambda \geq 0$ and $\eta = \frac{2}{\Lambda}$. The former situation is more common, with a convex loss function and an appropriate learning rate. Then the error is $o(C)t$, linear with t.
- **Case 3** ($\gamma > 1$): In this case, $\lambda < 0$ or $\eta > \frac{2}{\Lambda}$, where either the loss function is non-convex or the learning rate is too large. Then the error bound is exponential with t, making estimation ineffective.

Improving Robustness and Efficiency

In this section, we improve the basic estimator from two aspects: one is to improve the robustness of the method in non-convex case, and the other is to cut down the high cost due to computing Hessian matrix.

Layer-Wise Examination and Truncation

The analysis in Section reveals that the basic estimator can have a large error when the loss function is non-convex. The non-convex case is quite common in federated learning for deep learning tasks (Yu, Yang, and Zhu 2019; Haddadpour et al. 2019). We thus propose layer-wise examination and truncation (LWET for short).

Truncation. Our primary goal is to avoid an exponential error in Case 3, which results from the estimation of sequential influence. We consider a counterpart, where the sequential influence is completely omitted, *i.e.*, $\epsilon_t^{-c} = \mathbf{w}_t\left(\mathcal{C}_t \to \mathcal{C}_t \backslash \{c\}\right) - \mathbf{w}_t$ (we call it the *truncated estimator* for simplicity), and find that the error becomes independent of t, as shown in the following theorem.

Theorem 2. *With the sequential influence omitted, we get the bound of error as follows:*

$$\delta_t^{-c} \leq \gamma C + o(C).$$

Layer-wise Operation. However, after truncation, a new problem arises: the truncated estimator will always be $\mathbf{0}$ at round t as long as c is not one of the participants. For example, in a setting where 5 clients are selected each round with 100 clients in total, there will be 95 clients with FIP being $\mathbf{0}$ each round, which does not make sense. To ensure accuracy while retaining as much information as possible, it is necessary to find a happy medium between the basic estimator and the truncated estimator, which instead partially omits the sequential influence. To achieve this, we first introduce layer-wise operation. We calculate only parts of the Hessian matrix, with the interaction between different layers ignored. We take a convolutional neural network (CNN) for example. Supposing that the parameter $\mathbf{w}$ is composed of $\mathbf{w}_{(j)}$, $j = 1, 2, \ldots, 8$, corresponding to conv-layer 1, bias of conv-layer 1, conv-layer 2, bias of conv-layer 2, dense-layer 1, bias of dense-layer 1, dense-layer 2, bias of dense-layer 2, respectively, *i.e.*, $\mathbf{w} = [\mathbf{w}_{(1)}^T, \mathbf{w}_{(2)}^T, \mathbf{w}_{(3)}^T, \mathbf{w}_{(4)}^T, \mathbf{w}_{(5)}^T, \mathbf{w}_{(6)}^T, \mathbf{w}_{(7)}^T, \mathbf{w}_{(8)}^T]^T$. For each layer j, we see $\mathcal{L}(\mathbf{w})$ as function of $\mathbf{w}_{(j)}$ denoted by $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$. Rather than computing the entire Hessian matrix $\mathbf{H} = \nabla_w^2 \mathcal{L}(\mathbf{w})$, we only calculate $\mathbf{H}_{(j)} = \nabla_{w_{(j)}}^2 \mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ for each j, some smaller matrices on the diagonal of $\mathbf{H}$. Then the estimator in each layer j is updated independently:

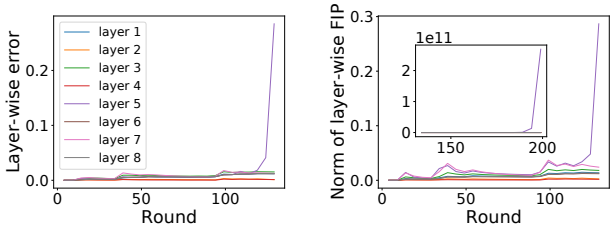
$$\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{M}_{t,(j)}^{-c} \epsilon_{t-1,(j)}^{-c} + \mathbf{w}_{t,(j)}\left(\mathcal{C}_t \to \mathcal{C}_t \backslash \{c\}\right) - \mathbf{w}_{t,(j)},$$

where $\mathbf{M}_{t,(j)}^{-c} = \sum_{k \in \mathcal{C}_t \backslash \{c\}} \frac{n_k}{N(\mathcal{C}_t \backslash \{c\})} \prod_{i=0}^{m-1} (\mathbf{I} - \eta \mathbf{H}_{t,i,(j)}^k)$. We can examine separately the property of the loss function $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ in each layer j, and use a layer-wise truncated estimator shown below only for layers in Case 3:

$$\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{w}_{t,(j)}\left(\mathcal{C}_t \to \mathcal{C}_t \backslash \{c\}\right) - \mathbf{w}_{t,(j)}. \tag{13}$$

Then we put together layer-wise FIPs to get the complete FIP, *i.e.*, $\epsilon_t^{-c} = [(\epsilon_{t,(1)}^{-c})^T, (\epsilon_{t,(2)}^{-c})^T \ldots]^T$.

Because the convexity and continuity of $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ vary among different layers, there are layers in Case 1 or Case 2 which still remain the sequential influence. Therefore the complete FIP is non-zero and contains much information even when $c \notin \mathcal{C}_t$. In an experiment conducted on a toy model (CNN 2 in Table 2), we found the exponential error only exists in the first fully-connected layer, which is the only layer where we need to apply a layer-wise truncated estimator, as shown in Fig. 1(a).



(a) (b)

Figure 1: The experimental result on a toy model (setting 2 in Table 2), a CNN without activation function. (a) The error and (b) the norm of estimated FIP both have a tendency to increase exponentially on layer 5, the first fully-connected layer. Inset: the norm of estimated FIP from round 135 to 200.

Examination. The next problem is how we can examine the property of loss function in each layer? We propose a mechanism which does not require any *prior knowledge*: in particular, we compare $\|\mathbf{M}_{t,(j)}^{-c} \epsilon_{t-1,(j)}^{-c}\|$ with $\|\epsilon_{t-1,(j)}^{-c}\|$. If $\|\mathbf{M}_{t,(j)}^{-c} \epsilon_{t-1,(j)}^{-c}\|$ is larger at a certain round r, then, in all of the following rounds, *i.e.*, for all $t \geq r$, a layer-wise truncated estimator in Equation 13 will be used in layer j.

This mechanism is aimed at examining a sufficient (but not necessary) condition for $\gamma > 1$. Please refer to the supplement for the reason. In addition, this examination also avoids the overflow of the estimator itself. The upper bound of ϵ_{t-1}^{-c} is also exponential with t in Case 3, which means

there is always a risk for it to overflow as t increases. This phenomenon can be observed in the experiment with the aforementioned toy model. As shown in Fig. 1(b), the norm of estimated FIP on the first fully-connected layer increases sharply around the 125-th round, and then achieves an order of 10^{11} at round 200, indicating the failure of the algorithm.

Combining the three strategies, we get LWET. Because real-world applications rarely satisfy the Identically and Independently Distributed (IID) assumption and are likely to be non-IID in many ways (Zhao et al. 2018; Li et al. 2020a,b; Yan et al. 2020), where clients vary in the data distribution and the property of the local loss function, a more fine-grained version of LWET is recommended in these cases. We can examine the relationship between $\| \left(\prod_{i=0}^{m-1} (\mathbf{I} - \eta \mathbf{H}_{t,i,(j)}^k) \right) \epsilon_{t,(j)}^{-c} \|$ and $\| \epsilon_{t,(j)}^{-c} \|$. If the former is larger at one round, then we drop the local sequential influence on k from that round on. Please refer to this algorithm in the supplement.

Low-Cost Hessian Approximation

Fisher information. Because the cross entropy loss is a negative log-likelihood, it is not difficult to obtain that $\mathbb{E}_{z \in \mathcal{D}'} \left[\nabla_w \mathcal{L}(\mathbf{w}^*, z) \nabla_w \mathcal{L}(\mathbf{w}^*, z)^T \right]$, where $\mathbf{w}^*$ is the true parameter (*i.e.*, the model distribution under $\mathbf{w}^*$ equals to the underlying distribution), is at the form of Fisher information. And according to one of the alternative definition of Fisher information (Friedman, Hastie, and Tibshirani 2001; Ly et al. 2017), it can also be written as $\mathbb{E}_{z \in \mathcal{D}'} [\nabla_w^2 \mathcal{L}(\mathbf{w}^*, z)]$. This means we can use $\nabla_w \mathcal{L}(\mathbf{w}, z) \nabla_w \mathcal{L}(\mathbf{w}, z)^T$ as an asymptotically unbiased estimation of $\nabla_w^2 \mathcal{L}(\mathbf{w}, z)$, since $\mathbf{w}$ gradually converges to $\mathbf{w}^*$ during the training.

Leveraging the fact that clients holds their own datasets, we let each client randomly select a given number, denoted by N_s, of gradients and use the empirical expectation of the outer product as an approximation to the Hessian, denoted by $\tilde{\mathbf{H}}_{t,i,(j)}^k$:

$$\mathbf{H}_{t,i,(j)}^k \approx \tilde{\mathbf{H}}_{t,i,(j)}^k \stackrel{\text{def}}{=} \frac{1}{N_s} \sum_{z \in \mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z) g(\mathbf{w}_{t,i,(j)}^k, z)^T$$

where $\mathcal{S}_{t,i}^k$ is the set of N_s samples randomly selected by client k at the i-th local iteration in round t, and $g(\mathbf{w}_{t,i,(j)}^k, z) = \nabla_{w_{(j)}} \mathcal{L}_{(j)}(\mathbf{w}_{t,i,(j)}^k, z)$.

Recursive computation. Simply introducing Fisher information cannot help cut down the cost, because the outer product is $O(p^2)$, and the $O(p^2)$ matrix-vector or $O(p^3)$ matrix-matrix multiplication still exists, where p is the size of the model. However, we can actually circumvent these needless calculations. With $\mathbf{H}_{t,i,(j)}^k$ approximated by $\tilde{\mathbf{H}}_{t,i,(j)}^k$, the estimator of local sequential influence is

$$\left(\prod_{i=0}^{m-1} \left(\mathbf{I} - \eta \frac{1}{N_s} \sum_{z \in \mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z) g(\mathbf{w}_{t,i,(j)}^k, z)^T \right) \right) \epsilon_{t-1,(j)}^{-c}. \tag{14}$$

	Server	Client	Comm.
Naive 1	K^2p^2	$m(np^2 + p^3)$	p^2
Naive 2	K^2mp^2	mnp^2	mp^2
Our method	K^2mN_sp	0	mN_sp

Table 1: Extra time complexity for the server, extra time complexity for each client, and extra communication complexity for each client. n is the size of the local dataset and $K = |\mathcal{C}|$. In a naive implementation, operations where the Hessian matrix is involved, can be taken either on the clients or the server, corresponding to naive implementations 1 and 2, respectively. For naive implementation 1, each client k has to compute $\prod_{i=0}^{m-1} (\mathbf{I} - \eta \mathbf{H}_{t,i}^k)$ locally, which requires matrix-matrix multiplications, and upload the result to the server. For naive implementation 2, each client k uploads $\mathbf{H}_{t,i}^k$ for all i, and then the server computes $\mathbf{M}_t^{-c} \epsilon_{t-1}^{-c}$, which can be implemented with only matrix-vector multiplications.

Instead of first calculating the cumprod on the left and then multiplying it by $\epsilon_{t-1,(j)}^{-c}$, the linear-cost method is based on a recursive computation: we initialize a vector $\sigma_{(j)}^k$ by $\sigma_{(j)}^k \leftarrow \epsilon_{t-1,(j)}^{-c}$, and then we update $\sigma_{(j)}^k$ using Equation 15 repeatedly for i from 0 to $m-1$:

$$\sigma_{(j)}^k \leftarrow \sigma_{(j)}^k - \frac{\eta}{N_s} \sum_{z \in \mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z) \left(g(\mathbf{w}_{t,i,(j)}^k, z)^T \sigma_{(j)}^k \right). \tag{15}$$

$\sigma_{(j)}^k$ produced in the final iteration is the value of Equation 14. Note that this method requires the computation to take place on the server, because only the server has $\epsilon_{t-1,(j)}^{-c}$, and the clients need to do nothing other than randomly select and upload a certain number of local gradients. We show the efficiency of our method by comparing it with naive implementations in Table 1.

Fundamental Experiments

In this section, we demonstrate two properties of our method: (1) LWET plays a vital role; and (2) Hessian approximation causes only a slight drop in the accuracy. Experiments are conducted on 64bit Ubuntu 18.04 LTS with four Intel i9-9900K CPUs and two NVIDIA RTX-2080TI GPUs, 200GB storage. We take "Leaf" (Caldas et al. 2018), a benchmarking framework for federated learning based on tensorflow. We evaluated our method on three settings, as shown in Table 2. We used the softmax function at the output layer and adopted the cross entropy as the loss function. In setting 1, the loss function is convex but not strongly convex, and therefore it is in Case 2 ($\gamma = 1$). In setting 2, although the toy model has no activation function, which makes it equivalent to a single-layer perceptron with convex loss function, results show that it is still in Case 3 ($\gamma > 1$) because the learning rate is too large. And in setting 3, the loss function is non-convex and is therefore in Case 3, too.

	Model	Dataset	Distribution	η	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{C}_t\rvert$	m	T	N_s
Setting 1	LogReg	Synthetic	Non-IID, Unbalanced	0.003	1000	10	5	1000	50
Setting 2	CNN 1	FEMNIST	IID, Balanced	0.03	50	5	2	500	50
Setting 3	CNN 2	FEMNIST	Non-IID, Unbalanced	0.02	100	10	2	2000	50

Table 2: Detailed configuration of the three different settings. The two datasets are described in Caldas et al. (2018). The logistic regression model is the original one in "Leaf". We made a little adjustment to the original CNN to create models with certain properties and scales; see details in the supplement.

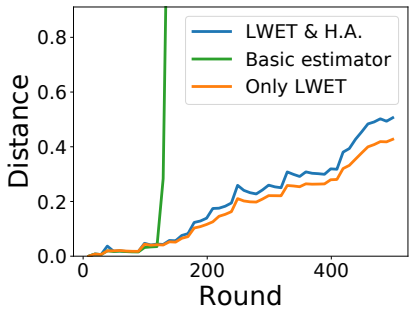


Figure 2: The distance between exact and estimated FIP in setting 2.

Influence on Parameter

We use four different methods to obtain ϵ_t^{-c}: (1) the basic estimator, (2) the estimator with only LWET, (3) the estimator with only Hessian approximation, and (4) the estimator with both LWET and Hessian approximation. We do not demonstrate results of all methods in each setting. In setting 1, we had tested all of the four methods, but found that the result produced with LWET is completely the same with that produced without LWET, which further validates that setting 1 is in Case 2. Therefore we only show two different results, the result based on exact Hessian and that based on approximated Hessian. In setting 2, we do not demonstrate the method with only Hessian approximation, because the basic estimator has already caused a terrible error, and the introduction of Hessian approximation will, no doubt, create an even larger error. In setting 3, we can only test the two methods that contain a Hessian approximation because the large model size and the limited resources on our device do not allow us to compute the exact Hessian.

We examine the error of the proposed method $\|\epsilon_t^{-c,*} - \epsilon_t^{-c}\|$, which is the distance between exact FIP and estimated FIP under L_2 norm. The exact FIP is obtained by conducting leave-one-out tests. We track the error of one randomly selected client's FIP and show the result in setting 2 in figure 2; results in the other two settings are provided as supplementary materials. Here we can see the necessity of LWET: without LWET there will be a sharp increase in the error at about the 120-th round (the error is caused by the first fully-connected layer as mentioned earlier in Fig. 1). Further taking a Hessian approximation only causes a slight increase in the error.

Influence on Loss

In this section, we give a more intuitive demonstration of the experimental results by mapping the high-dimensional FIP to a scalar, FIL. The exact FIL is obtained from the result of leave-one-out test.

By adding estimated FIP to the original global model, we get a estimation of the model trained with one client removed $\mathbf{w}_t + \epsilon_t^{-c}$. Then we test it on $\mathcal{D}_{test}$ and subtract from the result the loss of the original model to get the estimated FIL, *i.e.*, $\mathcal{L}(\mathbf{w}_t + \epsilon_t^{-c}, \mathcal{D}_{test}) - \mathcal{L}(\mathbf{w}_t, \mathcal{D}_{test})$.

We compare the estimated FIL with the exact FIL and show the results in Fig. 3. The correlation between the estimated and exact FIL is measured by Pearson's correlation coefficient, and we plot its variation with time in Figs 3(a), 3(b), and 3(c). We also visualize the correlation in the last round in Figs 3(d), 3(e), and 3(f). In particular, the proposed method achieves a Pearson correlation coefficient of 0.6200 at the last round in setting 3, the most difficult setting; in setting 1 Pearson correlation coefficient is 0.9857 and in setting 2 it is 0.7957.

Extended Experiments

In this section, we use FIL and FIA for client valuation and client cleansing, despite that we believe there are more potential applications based on other types of influence given by FIP, for example, the client-level influence on the prediction results.

Influence on model performance can be used as a reasonable metric to determine a client's value. We conduct the following experiment: we make an observation at clients' influence at a certain round, remove some of the clients with highest/lowest influence, from the training set, and then continue the learning process. The experiment is repeated with different fractions of clients removed and the performance of the final global model is recorded each time. As can be seen from Fig. 4, removing valuable clients (those with high FIL or low FIA) greatly degrades the model performance, which indicates the importance of these clients. In contrast, removing least valuable clients (those with low FIL or high FIA) improves the model performance. And the curve of randomly removing falls between the other two curves.

The results of removing least valuable clients elicits an application, which we call client cleansing. It is a little different from the data cleansing in Hara, Nitanda, and Maehara (2019). Data cleansing is conducted by retraining the model with a subset of data removed. In the setting of federated learning, as we mentioned before, it does not make

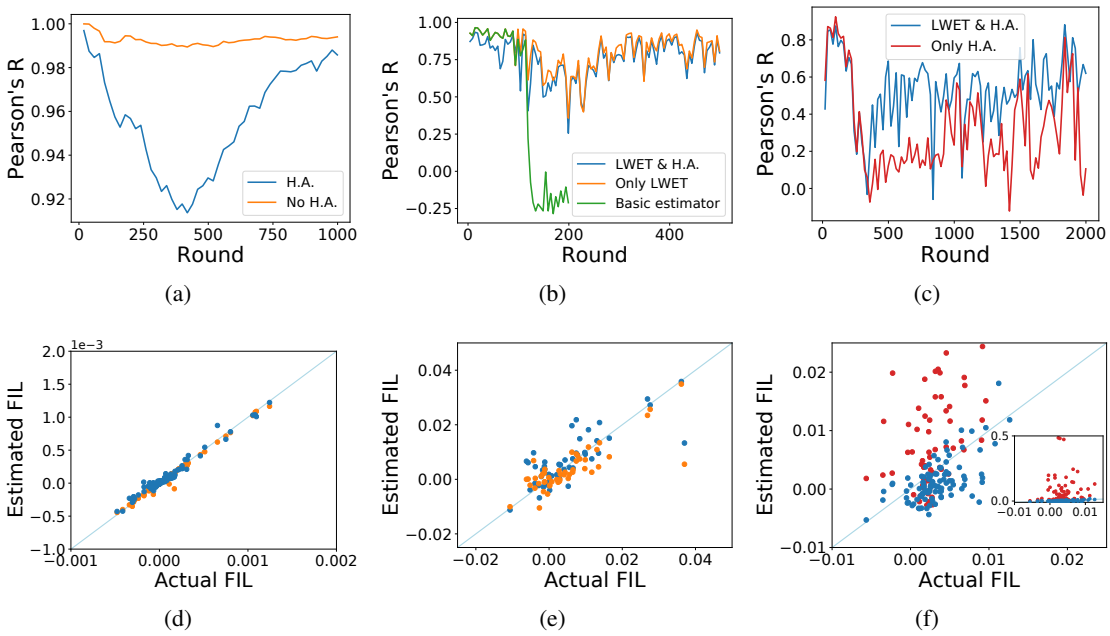


Figure 3: (a)(b)(c) The change of Pearson coefficient over rounds in setting 1, 2 and 3. (d)(e)(f) Estimated FIL vs. actual FIL in settings 1, 2 and 3. The inset in (f) shows the results with a wider y-axis range. In setting 1, we counted 200 clients that were randomly selected from the 1000 clients. In settings 2 and 3, we counted all the clients (50 and 100 clients, respectively).

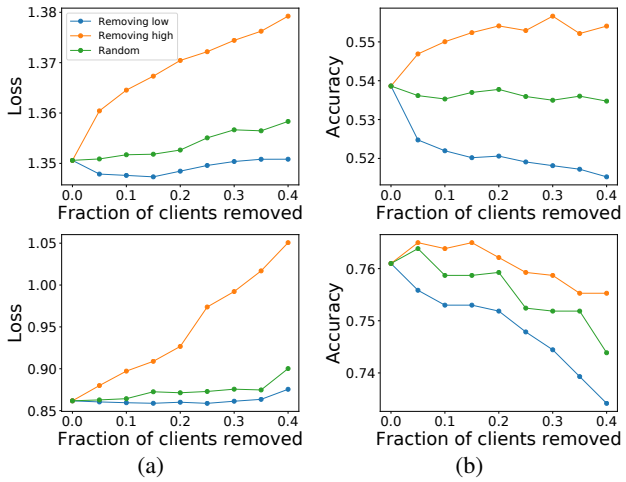


Figure 4: We remove clients from the client set at a certain round x in three different ways: from those with lowest influence, from those with highest influence, randomly. (a) We use influence on loss and record the corresponding change in loss. (b) We use influence on accuracy and record the corresponding change in accuracy. Top: The results in setting 1 with $x = 700$. Bottom: The results in setting 3 with $x = 1500$.

sense to retrain the model in most real-world scenarios. Therefore, we conduct client cleansing by removing a subset of clients at a certain round and then continue the training. By properly selecting the clients to be removed we can effectively improve the final model. In setting 1, the accuracy is increased from 53.86% to 55.66% by removing 30% of the clients from the client set at round 700. In setting 3, the accuracy is increased from 76.10% to 76.50% by removing 5% of the clients from the client set at round 1500.

Conclusion

In this paper, we proposed Fed-Influence as a new metric of clients' influence on the global model in federated learning, inspired by the classical statistics notion of influence in centralized learning. The differences between the centralized learning and federated learning create challenges in computing this new metric. To develop an efficient implementation, we first proposed a basic estimator of individual client's influence on parameter, then further revised the estimator and established an algorithm with both robustness and linear cost. It is worth noting that the proposed method is accurate without assumption on convexity or data identity, which is validated by the empirical results on different settings. We also demonstrated how Fed-Influence helps evaluate clients and improve the model performance through client cleansing. Our work provides a quantitative insight into the relationship between individual clients and the global model, we envision which further enhancing the accountability of federated learning.

Acknowledgements

This work was supported in part by National Key R&D Program of China No. 2019YFB2102200, in part by China NSF grant No. 62025204, 61972252, 61972254, 61672348, and 61672353, in part by Joint Scientific Research Foundation of the State Education Ministry No. 6141A02033702, and in part by Alibaba Group through Alibaba Innovation Research Program. The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government. Z. Zheng is the corresponding author.

References

Caldas, S.; Wu, P.; Li, T.; Konecný, J.; McMahan, H. B.; Smith, V.; and Talwalkar, A. 2018. LEAF: A Benchmark for Federated Settings. *arXiv preprint arXiv:1812.01097* .

Cook, R.; and Weisberg, S. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics* 495–508.

Cook, R. D. 1977. Detection of influential observation in linear regression. *Technometrics* 15–18.

Cook, R. D.; and Weisberg, S. 1982. *Residuals and influence in regression*. New York: Chapman and Hall.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The elements of statistical learning*. Springer series in statistics New York.

Haddadpour, F.; Kamani, M. M.; Mahdavi, M.; and Cadambe, V. R. 2019. Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization. In *Proc. of NeurIPS*, 11080–11092.

Hamer, J.; Mohri, M.; and Suresh, A. T. 2020. FedBoost: A Communication-Efficient Algorithm for Federated Learning. In *Proc. of ICML*, 3973–3983.

Hampel, F. R. 1974. The influence curve and its role in robust estimation. *Journal of the american statistical association* 383–393.

Hara, S.; Nitanda, A.; and Maehara, T. 2019. Data Cleansing for Models Trained with SGD. In *Proc. of NeurIPS*, 4215–4224.

Jaeckel, L. 1972. *The infinitesimal jackknife*. Unpublished memorandum, Bell Telephone Laboratories.

Kairouz, P.; McMahan, H. B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A. N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* .

Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S. J.; Stich, S. U.; and Suresh, A. T. 2019. Scaffold: Stochastic controlled averaging for on-device federated learning. *arXiv preprint arXiv:1910.06378* .

Khanna, R.; Kim, B.; Ghosh, J.; and Koyejo, S. 2019. Interpreting Black Box Predictions using Fisher Kernels. In *Proc. of AISTATS*, 3382–3390.

Koh, P. W.; Ang, K.; Teo, H. H. K.; and Liang, P. 2019. On the Accuracy of Influence Functions for Measuring Group Effects. In *Proc. of NeurIPS*, 5255–5265.

Koh, P. W.; and Liang, P. 2017. Understanding Black-box Predictions via Influence Functions. In *Proc. of ICML*, 1885–1894.

Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2020a. Federated Optimization in Heterogeneous Networks. In *Proc. of MLSys*.

Li, X.; Huang, K.; Yang, W.; Wang, S.; and Zhang, Z. 2020b. On the Convergence of FedAvg on Non-IID Data. In *Proc. of ICLR*.

Ly, A.; Marsman, M.; Verhagen, J.; Grasman, R. P.; and Wagenmakers, E.-J. 2017. A tutorial on Fisher information. *Journal of Mathematical Psychology* 40–55.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proc. of AISTATS*, 1273–1282.

Rothchild, D.; Panda, A.; Ullah, E.; Ivkin, N.; Stoica, I.; Braverman, V.; Gonzalez, J.; and Arora, R. 2020. Fetchsgd: Communication-efficient federated learning with sketching. In *Proc. of ICML*, 8253–8265.

Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; and Poor, H. V. 2020. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Proc. of NeurIPS* .

Yan, Y.; Niu, C.; Ding, Y.; Zheng, Z.; Wu, F.; Chen, G.; Tang, S.; and Wu, Z. 2020. Distributed Non-Convex Optimization with Sublinear Speedup under Intermittent Client Availability. *arXiv preprint arXiv:2002.07399* .

Yu, H.; Yang, S.; and Zhu, S. 2019. Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning. In *Proc. of AAAI*, 5693–5700.

Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; and Chandra, V. 2018. Federated Learning with Non-IID Data. *arXiv preprint arXiv:1806.00582* .