Accelerating Federated Learning with Split Learning on Locally Generated Losses

Dong-Jun Han [1] **Hasnain Irshad Bhatti** [1] **Jungmoon Lee** [1] **Jaekyun Moon** [1]

Abstract

Federated learning (FL) operates based on model exchanges between the server and the clients, and suffers from significant communication as well as client-side computation burden. While emerging split learning (SL) solutions can reduce the client-side computation burden by splitting the model architecture, SL-based ideas still require significant time delay and communication burden for transmitting the forward activations and backward gradients at every global round. In this paper, we propose a new direction to FL/SL based on updating the client/server-side models in parallel, via local-loss-based training specifically geared to split learning. The parallel training of split models substantially *shortens latency* while *obviating server-to-clients communication*. We provide latency analysis that leads to optimal model cut as well as general guidelines for splitting the model. We also provide a theoretical analysis for guaranteeing convergence of our method. Extensive experimental results indicate that our scheme has significant communication and latency advantages over existing FL and SL ideas.

1. Introduction

Federated learning (FL) (McMahan et al., 2017; Konečnỳ et al., 2016b) is being regarded as a promising direction for distributed learning, as it enables clients to collaboratively train a global model without directly uploading their privacy-sensitive data to the server. However, in FL, each client should repeatedly download the entire model from the server, update the model, and upload it back to the server. This training process causes significant computation/communication burdens especially with deep neural networks having large numbers of model parameters. Moreover, when the computing powers and the transmission rates of the clients are low (e.g., mobile/IoT devices), FL requires significant computation/communication delays. These issues can limit the application of FL in practical scenarios aiming to train a *large-scale model* using local data of clients given *low computing powers* and *low transmission rates*.

Split learning (SL) (Gupta & Raskar, 2018; Vepakomma et al., 2018; Thapa et al., 2020) is another recent approach for this setup, which can reduce the computation burden at the clients by splitting the model $\mathbf{w}$ into two parts: the first few layers (client-side model $\mathbf{w}_C$) are allocated to the clients, and the remaining layers (server-side model $\mathbf{w}_S$) are allocated to the server. Since each client only need to train the first few layers of the model, the computational burden at each client is reduced compared to FL.

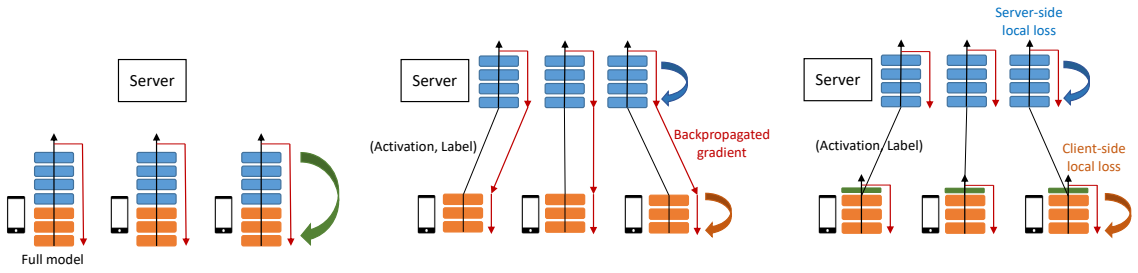
However, existing SL-based ideas still have two critical issues in terms of latency and communication efficiency. First, existing SL solutions still require significant time delay, since each participating client should wait for the backpropagated gradients from the server in order to update its model. Moreover, the communication burden can still be substantial for transmitting the forward/backward signals via uplink/downlink communications at each global round.

Contributions: In this paper, we propose a fast and communication-efficient solution that provides a new direction to federated/split learning, by addressing the high latency requirement and high communication resource requirement of current FL and SL-based approaches. Motivated by the idea of local-loss-based training (Nøkland & Eidnes, 2019; Belilovsky et al., 2020), instead of considering the conventional loss function that is computed at the output of the model $\mathbf{w}$, we introduce alternative local loss functions specifically geared to the split learning setup. We develop an algorithm where the client-side models can be updated without receiving the backpropagated signals from the server, significantly improving latency and communication efficiency. Fig. 1 compares our idea with FL and the state-of-the-art SL approach, termed SplitFed (Thapa et al., 2020). Our main contributions are summarized as follows:

- We propose a **new federated split learning algorithm** that addresses the latency and communication

[1]School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea.

This work was presented at the International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21).



(a) Federated Learning (FL) (b) SplitFed (c) Proposed

Figure 1. Model update process of FL, SplitFed and our idea. FL suffers from large time delay and communication/computation burdens for exchanging/updating the full model. Although SplitFed can reduce the client-side computation burden, it still requires large time delay since the clients should wait for the backpropagated signals from the server in order to update their model. The communication burden can be also large for transmitting the forward activations and backward gradients at every global round. The proposed idea enables fast and communication-efficient learning by parallelizing client/server-side model updates, via local-loss-based training geared to split learning.

efficiency issues of current FL and SL approaches, via local-loss-based training geared to split learning.

- We provide **latency analysis** and provide an optimal solution on splitting the model to minimize the latency. We also provide theoretical analysis to **guarantee convergence** of our scheme.
- Experimental results show that our approach **outperforms existing FL and SL-based ideas** in practice where clients having low computing powers and low transmission rates collaborate to train a global model.

2. Basic Setup and Related Works

Consider a system with a single server and N clients having their own local/private data. FL and SL are the recent ideas that aim to train a model in this setup. The goal is generally to find $\mathbf{w}^*$ that minimizes the loss function defined as $F(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} F_k(\mathbf{w})$. Here, $F_k(\mathbf{w})$ is the loss function at client k defined as $F_k(\mathbf{w}) = \frac{1}{|D_k|}\sum_{x\in D_k} \ell(x;\mathbf{w})$ where D_k is the dataset of client k and $\ell(x;\mathbf{w})$ is the loss computed by the model $\mathbf{w}$ and the data sample x.

Federated Learning: In FL (McMahan et al., 2017; Konečnỳ et al., 2016b;a; Li et al., 2020), the above problem is solved via repeated model download at the clients and aggregation at the server. At every global round t, the server randomly selects A_t, a set of K clients participating in FL in this round. Each client $k \in A_t$ downloads the model $\mathbf{w}^t$ from the server and performs local update to obtain $\mathbf{w}_k^{t+1}$. The server aggregates the models from all clients to obtain $\mathbf{w}^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{w}_k^{t+1}$ and moves on to the next round. However, when training a large-scale model, FL causes significant communication burden between the server and the clients for model exchange, and considerable computation burden for training the model at low-powered clients.

Split Learning: SL (Gupta & Raskar, 2018; Vepakomma et al., 2018; Thapa et al., 2020) is another direction to train the model in this setup while reducing the computation burden at the clients compared to FL. The basic idea of SL approaches is to split the model $\mathbf{w}$ into two modules as $\mathbf{w} = [\mathbf{w}_C, \mathbf{w}_S]$. The first few layers $\mathbf{w}_C$ correspond to the client-side model, and the remaining layers $\mathbf{w}_S$ correspond to the server-side model. Among existing SL ideas, SplitFed (Thapa et al., 2020) achieves the state-of-the-art performance by parallelizing SL. At each global round t, the server randomly selects a set A_t that consists of K participating clients in the current round. Each client $k \in A_t$ downloads the client-side model $\mathbf{w}_C^t$ from the server, performs forward propagation with its local data, and sends the output and the corresponding labels to the server. The server proceeds forward propagation, computes the loss, and performs backpropagation to update the server-side models in parallel, as in Fig. 1(b). Now the server transmits the corresponding backpropagated signal to each client. Each client k can update the model by proceeding backpropagation to obtain $\mathbf{w}_{C,k}^t$. Finally, the server aggregates the updated models from all clients to obtain $\mathbf{w}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{w}_{C,k}^{t+1}$. The server-side models are also aggregated to obtain $\mathbf{w}_S^{t+1}$. However, although SplitFed can reduce the client-side computation burden compared to FL, existing SL-based ideas still have issues in terms of latency and communication efficiency; all participating clients should receive the backpropagated signals from the server in order to update their models, which require significant time delay and communication resources at every global round.

Local-loss-based training: Local-loss-based training represents schemes that aim to train a model using local error signals (local loss functions) instead of using the conventional global loss function. Auxiliary networks are utilized to compute the local error signals. By utilizing the local losses, layer-wise training (Nøkland & Eidnes, 2019; Belilovsky et al., 2019) or module-wise training (Belilovsky et al., 2020; Laskin et al., 2020) is possible without receiving the backpropagated signals from the previous layer or module. While existing works on local-loss-based training consider a centralized setup (i.e., central node having the entire dataset and the model), in this paper, we specifically focus on a distributed setup (i.e., data distributed across the clients and model splitted between the client/server) and

propose a local-loss-based training method highly tailored to split learning. Theoretical and experimental results indicate that our new algorithm can provide a new direction to federated/split learning via local-loss-based training.

3. Proposed Algorithm

In this section, we describe our algorithm which addresses the latency and communication burden issues of FL and SL. As in SL approaches, we first split the model $\mathbf{w}$ into the client-side model $\mathbf{w}_C$ and the server-side model $\mathbf{w}_S$. Our goal is to obtain the optimal model $\mathbf{w}^* = [\mathbf{w}_C^*, \mathbf{w}_S^*]$.

3.1. Local Loss Functions

Instead of considering the conventional loss function that is computed at the output of the model $\mathbf{w}$, our idea is to consider two different *local loss functions* specifically geared to the split learning setup.

Client-side local loss function: We first describe the client-side local loss function. We introduce an auxiliary network $\mathbf{a}_C$ to make a prediction at the client-side and then compute the local loss. Here, the auxiliary network $\mathbf{a}_C$ is the extra layers connected to the client-side model $\mathbf{w}_C$; the output of $\mathbf{w}_C$ becomes the input of $\mathbf{a}_C$. Both convolutional neural networks or multilayer perceptrons (MLP) can be utilized for the auxiliary network $\mathbf{a}_C$, which is our design choice. In this work, we adopt a MLP for $\mathbf{a}_C$ as in (Belilovsky et al., 2020; Laskin et al., 2020) to match the dimension between the output of $\mathbf{a}_C$ and the target label. We show later in Section 5 that only a very small size auxiliary network is sufficient (0.1% of the entire model size $|\mathbf{w}|$) to achieve the state-of-the-art performance. Our goal is to find $\mathbf{w}_C^*$ and $\mathbf{a}_C^*$ that minimizes the client-side loss function $F_C(\cdot)$ which is the average of local loss functions of all clients:

$$\min_{\mathbf{w}_C, \mathbf{a}_C} F_C(\mathbf{w}_C) = \min_{\mathbf{w}_C, \mathbf{a}_C} \frac{1}{N} \sum_{k=1}^{N} F_{C,k}(\mathbf{w}_C, \mathbf{a}_C), \quad (1)$$

where $F_{C,k}(\cdot, \cdot)$ is the local loss function of the client-side model at client k, defined as $F_{C,k}(\mathbf{w}_C, \mathbf{a}_C) = \frac{1}{|D_k|} \sum_{x \in D_k} \ell(x; \mathbf{w}_C, \mathbf{a}_C)$. Here, $\ell(x; \mathbf{w}_C, \mathbf{a}_C)$ is the cross-entropy loss computed using input x, client-side model $\mathbf{w}_C$ and the auxiliary network $\mathbf{a}_C$.

Server-side local loss function: The local loss function of the server-side model $\mathbf{w}_S$ is defined based on the optimal client-side model $\mathbf{w}_C^*$ defined in (1). We would like to find $\mathbf{w}_S^*$ that minimizes the server-side loss function $F_S(\cdot)$:

$$\min_{\mathbf{w}_S} F_S(\mathbf{w}_S) = \min_{\mathbf{w}_S} \frac{1}{N} \sum_{k=1}^{N} F_{S,k}(\mathbf{w}_S, \mathbf{w}_C^*), \quad (2)$$

where $F_{S,k}(\cdot, \cdot)$ is the local loss function of the server-side model corresponding to client k, defined as $F_{S,k}(\mathbf{w}_S, \mathbf{w}_C^*) = \frac{1}{|D_k|} \sum_{x \in D_k} \ell(g_{\mathbf{w}_C^*}(x); \mathbf{w}_S)$. Here, note that the input of $\ell(g_{\mathbf{w}_C^*}(x); \mathbf{w}_S)$ is $g_{\mathbf{w}_C^*}(x)$, which is defined as the output of the model $\mathbf{w}_C^*$ given the input x. We also note that the auxiliary network is not necessary at the server-side; the loss can be directly computed without the auxiliary network after finishing forward propagation of the server-side model.

3.2. Algorithm Description

Now we describe our algorithm to solve the above problem. Starting from the initial model $\mathbf{w}^0 = [\mathbf{w}_C^0, \mathbf{w}_S^0]$, we obtain $\mathbf{w}^T = [\mathbf{w}_C^T, \mathbf{w}_S^T]$ after T global rounds. As in (Thapa et al., 2020), we consider two different servers, the main server and the fed server. The main server updates $\mathbf{w}_S$ while the fed server only aggregates the models sent from the clients via FedAvg. In the beginning of each global round t, the server randomly selects the participating group A_t with K clients. Now we have the following four steps with steps 3 and 4 working in parallel.

Step 1 (Model download): At a specific global round t, each client $k \in A_t$ downloads $\mathbf{w}_C^t$ and $\mathbf{a}_C^t$ from the fed server and lets $\mathbf{w}_{C,k}^t = \mathbf{w}_C^t$, $\mathbf{a}_{C,k}^t = \mathbf{a}_C^t$.

Step 2 (Forward propagation and upload): Based on the downloaded model $\mathbf{w}_{C,k}^t$, each client k performs forward propagation for all data samples $x \in \tilde{D}_k$ in a specific mini-batch $\tilde{D}_k \subset D_k$. Specifically, client k obtains $g_{\mathbf{w}_{C,k}^t}(x)$ for all $x \in \tilde{D}_k$, which is the output of the model $\mathbf{w}_{C,k}^t$ given an input data x. Then each client k uploads $g_{\mathbf{w}_{C,k}^t}(x)$ to the main server for all $x \in \tilde{D}_k$.

Step 3 (Client-side model update and aggregation): Now based on the local loss function, each client k updates its model $\mathbf{w}_{C,k}^t$ and the auxiliary network $\mathbf{a}_{C,k}^t$:

$$\mathbf{w}_{C,k}^{t+1} = \mathbf{w}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{w}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t) \quad (3)$$

$$\mathbf{a}_{C,k}^{t+1} = \mathbf{a}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{a}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t) \quad (4)$$

where η_t is the learning rate at round t and $\tilde{\nabla} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$ is the derivative for a specific mini-batch, i.e., $\tilde{\nabla} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t) = \frac{1}{|\tilde{D}_k|} \sum_{x \in \tilde{D}_k} \nabla \ell(x; \mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$. After the model update process, the fed server aggregates the client-side models as $\mathbf{w}_C^{t+1} = \frac{1}{K} \sum_{k \in A_t} \mathbf{w}_{C,k}^{t+1}$. The auxiliary networks are also aggregated as $\mathbf{a}_C^{t+1} = \frac{1}{K} \sum_{k \in A_t} \mathbf{a}_{C,k}^{t+1}$.

Step 4 (Server-side model update and aggregation, working in parallel with step 3): While the client-side models are being updated using the local errors, in parallel with step 3, the main server also updates the server-side model. Based on $g_{\mathbf{w}_{C,k}^t}^t(x)$ received from each client k in step 2, the main server performs model update according to

$$\mathbf{w}_{S,k}^{t+1} = \mathbf{w}_S^t - \eta_t \tilde{\nabla} F_{S,k}(\mathbf{w}_S^t, \mathbf{w}_C^t) \quad (5)$$

Table 1. Computation load (per client), total communication load, and latency required for one global round.

Methods	Computation	Communication	Latency
FL	$\|D\|\|\mathbf{w}\|$	$2\|\mathbf{w}\|K$	$\frac{2\|\mathbf{w}\|K}{R}+\frac{\|D\|\|\mathbf{w}\|}{P_C}$
SplitFed	$\alpha\|D\|\|\mathbf{w}\|$	$(2q\|D\|+2\alpha\|\mathbf{w}\|)K$	$\frac{(2q\|D\|+2\alpha\|\mathbf{w}\|)K}{R}+\frac{\alpha\|D\|\|\mathbf{w}\|}{P_C}+\frac{(1-\alpha)\|D\|\|\mathbf{w}\|K}{P_S}$
Ours	$\alpha\|D\|\|\mathbf{w}\|$	$(q\|D\|+2\alpha\|\mathbf{w}\|)K$	$\frac{(q\|D\|+\alpha\|\mathbf{w}\|)K}{R}+\frac{\alpha\beta\|D\|\|\mathbf{w}\|}{P_C}+\max\left(\frac{\alpha\|\mathbf{w}\|K}{R}+\frac{\alpha(1-\beta)\|D\|\|\mathbf{w}\|}{P_C},\frac{(1-\alpha)\|D\|\|\mathbf{w}\|K}{P_S}\right)$

in parallel for all $k \in A_t$, where $\tilde{\nabla} F_{S,k}(\mathbf{w}_S^t, \mathbf{w}_C^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(g_{\mathbf{w}_{C,k}^t}(x);\mathbf{w}_S^t)$. Now the server-side models are aggregated as $\mathbf{w}_S^{t+1} = \frac{1}{K}\sum_{k\in A_t}\mathbf{w}_{S,k}^{t+1}$.

After repeating the overall procedure for T global rounds, we obtain $\mathbf{w}^T = [\mathbf{w}_C^T, \mathbf{w}_S^T]$. This final model is utilized at inference stage to make predictions.

3.3. Latency Analysis

Notations: Let $|\mathbf{w}|$ be the number of model parameters of $\mathbf{w}$, and α be the fraction of model parameters in $\mathbf{w}_C$: we have $|\mathbf{w}_C| = \alpha|\mathbf{w}|$ and $|\mathbf{w}_S| = (1-\alpha)|\mathbf{w}|$. We note that the auxiliary network is our design choice which can be made to have a significantly small number of parameters, i.e., $|\mathbf{a}| \ll |\mathbf{w}|$. Hence, we neglect the effect of the auxiliary network for latency analysis. We show later in Section 5 that the state-of-the-art performance can be achieved with negligible size of auxiliary network (0.1% of the entire model size $|\mathbf{w}|$). For analysis, we assume that all layers have the same size of q. Let P_C and P_S be the computing powers at the client and the server, respectively. For a given local dataset size $|D|$, model size $|\mathbf{w}|$ and computing power P, the required time for updating the model for one epoch is assumed to be $\frac{|D||\mathbf{w}|}{P}$. Here, we assume that the required time for the forward propagation is $\frac{\beta|D||\mathbf{w}|}{P}$ while the required time for the backpropagation is $\frac{(1-\beta)|D||\mathbf{w}|}{P}$. Finally, given a single client, we let both the uplink transmission rate from the client to server and the downlink transmission rate from the server to client as R. When K clients are communicating with the server simultaneously, the transmission rate of each client reduces to $\frac{R}{K}$.

Latency: In the beginning of each global round of our scheme, K clients simultaneously download $\mathbf{w}_C$ from the fed server, which requires latency of $\frac{\alpha|\mathbf{w}|K}{R}$. The forward propagation and transmitting the output to the main server requires additional latency of $\frac{\alpha\beta|D||\mathbf{w}|}{P_C}+\frac{q|D|K}{R}$. Now in parallel, each client updates its model and sends it to the fed server for aggregation, while the main server updates the server-side model. The latency is determined by the maximum value of these two, which leads to additional delay of $\max\left(\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C},\frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}\right)$. Table 1 compares various metrics of different methods.

Optimal splitting: Now we have the following question: how should we split the model to minimize the latency? In other words, what is the optimal α? The following theorem provides a guideline on splitting the model.

Theorem 1 *If $P_S \le (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, optimal α that minimizes the latency of our scheme is*

$$\alpha^* = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}. \tag{6}$$

Otherwise, the latency is an increasing function of α.

Theorem 1 states that if the computing power of the main server P_S is smaller than $(\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, one can select an appropriate α^* as (6) to minimize the latency. Here, the optimal α of (6) decreases with decreasing client-side computing power P_C, since larger latency is required for updating the client-side model with a smaller P_C. Moreover, α^* decreases with decreasing transmission rate R, since exchanging the model parameters between the server and the clients requires more latency with a smaller R. If P_S is larger than the threshold $(\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, it is beneficial to assign as many layers as possible to the server.

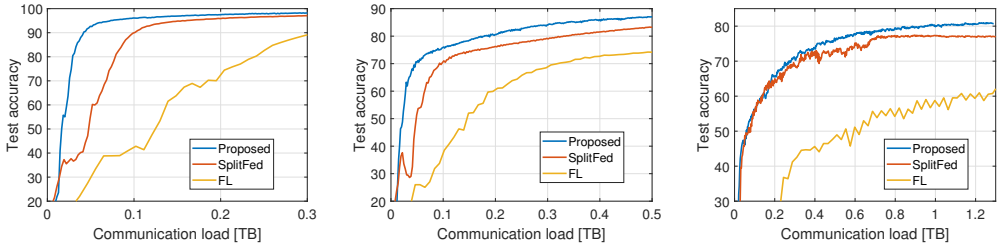
4. Convergence Analysis

In this section, we provide the convergence behavior of our scheme on non-convex loss functions with the following standard assumptions in FL (Li et al., 2019) and local-loss-based training (Belilovsky et al., 2020).

Assumption 1 *The client-side and server-side loss functions are L-smooth, i.e., $\|\nabla F_C(\mathbf{w}) - \nabla F_C(\mathbf{v})\| \le L\|\mathbf{w}-\mathbf{v}\|$, $\|\nabla F_S(\mathbf{w}) - \nabla F_S(\mathbf{v})\| \le L\|\mathbf{w}-\mathbf{v}\|$ hold for all $\mathbf{w}$, $\mathbf{v}$.*

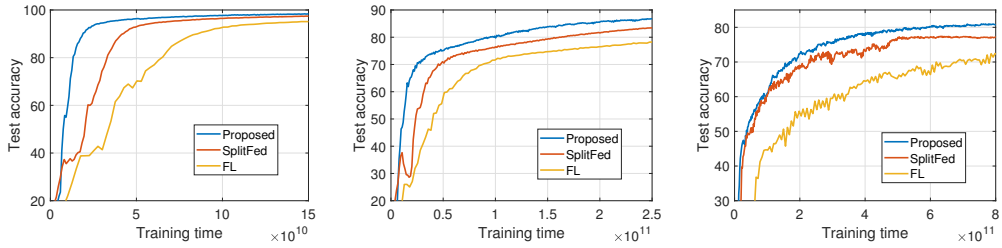
Assumption 2 *The second moment of the stochastic gradient is upper bounded, i.e., there exists G_1 such that $\|\nabla\ell(x;\mathbf{w})\|^2 \le G_1$ holds for any data sample $x \in \cup_{k=1}^N D_k$ and any $\mathbf{w}$. Similarly, considering the server-side loss, we assume that there exists G_2 such that $\|\nabla\ell(g_{\mathbf{w}_C^t}(x);\mathbf{w})\|^2 \le G_2$ holds for any data sample $x \in \cup_{k=1}^N D_k$ and for any t.*

Note that in each global round t, the output distribution of a specific client-side model after forward propagation (which is the input distribution of the server-side model) is determined by $\mathbf{w}_{C,k}^t$ and D_k. We let $z_{C,k}^t = g_{\mathbf{w}_{C,k}^t}(x)$ be the output of the k-th client-side model at global round t, following the probability distribution of $p_{C,k}^t(z)$. Here $p_{C,k}^t(z)$ is time-varying, and we let $p_{C,k}^*(z)$ be the output distribution of the k-th client-side model with $\mathbf{w}_C^*$ and D_k.



(a) MNIST (b) FMNIST (c) CIFAR-10

Figure 2. Test accuracy versus communication load.



(a) MNIST (b) FMNIST (c) CIFAR-10

Figure 3. Test accuracy versus training time.

We also define the distance between these two distributions as $d^t_{C,k} = \int \|p^t_{C,k}(z) - p^*_{C,k}(z)\|dz$. Now we provide our main theorem which shows the convergence behaviors of the client/server-side models.

Theorem 2 *Suppose Assumptions 1 and 2 hold. Let* $\Gamma_T = \sum_{t=0}^{T-1} \eta_t$. *After running the proposed algorithm for* T *global rounds, the client-side model converges as*

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_C(\mathbf{w}^t_C)||^2] \leq \frac{4(F_C(\mathbf{w}^0_C) - F_C(\mathbf{w}^*_C))}{3\Gamma_T} + \frac{G_1 L}{2}\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^2, \tag{7}$$

Moreover, the server-side model converges as

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}^t_S)||^2] \leq \frac{4(F_S(\mathbf{w}^0_S) - F_S(\mathbf{w}^*_S))}{3\Gamma_T} + G_2\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t \frac{1}{N}\sum_{k=1}^{N} d^t_{C,k} + \frac{L}{2}\eta_t^2\right). \tag{8}$$

Consider a diminishing step sizes $\eta_t = \frac{\eta_0}{1+t}$ which satisfy $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$. Then, as in the results of (Belilovsky et al., 2020), our algorithms converges to a stationary point: it can be seen from (7) that the right-hand side converges to zero as T grows. Regarding the server-side model, it can be seen from (8) that the sequence of expected gradient norm $\mathbb{E}[||\nabla F_S(\mathbf{w}^t_S)||^2]$ accumulates around 0 as $\inf_{t \leq T-1} \mathbb{E}[||\nabla F_S(\mathbf{w}^t_S)||^2] \leq \mathcal{O}(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \frac{1}{N}\sum_{k=1}^{N} d^t_{C,k})$. This is in the same form of the result in (Belilovsky et al., 2020) which considers the local-loss-based training method in a centralized setup. The difference here is that our rate $\mathcal{O}(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \frac{1}{N}\sum_{k=1}^{N} d^t_{C,k})$ is expressed as the average of distance $d^t_{C,k}$ of all clients, since we consider a distributed setup with multiple clients.

5. Experiments

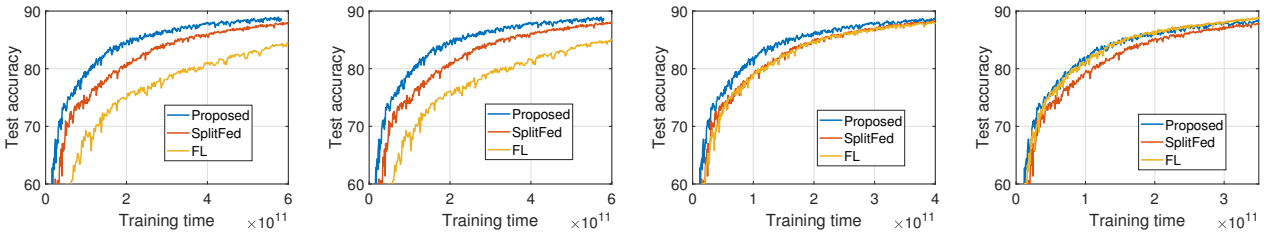
We validate our algorithm on MNIST (LeCun et al., 1998), FMNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky et al., 2009). For MNIST and FMNIST, we utilized a CNN having 5 convolutional layers and 3 fully connected (FC) layers as in AlexNet. The number of model parameters is $|\mathbf{w}| = 3,868,170$. For CIFAR-10, we utilized VGG-11 with $|\mathbf{w}| = 9,231,114$.

Data distribution: We distribute the training set of each dataset to the clients for training, and utilized the original test set of each dataset to evaluate the performance of the global model. We consider a system with $N = 1000$ clients. Hence, each client has 60 data samples for MNIST and FMNIST, and 50 data samples for CIFAR-10. Here, we consider two different data distribution setups, IID and non-IID setups. In an IID setup, data samples from each class is equally distributed across all $N = 1000$ clients in the system. Hence, each client has all 10 classes in its local dataset. In a non-IID setup, similar to the data distribution method in (McMahan et al., 2017), the training set is first divided into 5000 shards (12 data samples in each shard for MNIST/FMNIST, and 10 data samples in each shard for CIFAR-10). Then we allocate 5 shards to each client to model the non-IID scenario.

Model splitting and auxiliary network: We compare our result with FL (McMahan et al., 2017) and SplitFed (Thapa et al., 2020). In FL, the entire model $\mathbf{w}$ is updated at each

Table 2. Performance of different schemes at a specific time in Fig. 3.

	MNIST		FMNIST		CIFAR-10	
Methods	IID	Non-IID	IID	Non-IID	IID	Non-IID
FL	92.80%	89.02%	78.21%	75.77%	72.44%	62.84%
SplitFed	96.47%	95.47%	83.42%	82.44%	77.06%	75.02%
Proposed	**97.73%**	**97.01%**	**86.70%**	**85.74%**	**80.72%**	**78.91%**



(a) $P_C = 1, R = 1$ (b) $P_C = 10, R = 1$ (c) $P_C = 1, R = 4$ (d) $P_C = 10, R = 4$

Figure 4. Effect of client-side computing power P_C and transmission rate R. FMNIST is utilized in a non-IID setup. Our scheme is beneficial especially when the clients have relatively small computing powers and small transmission rates (e.g., mobile/IoT devices).

client and sent to the server for aggregation via FedAvg. In SplitFed and the proposed method, the model $\mathbf{w}$ is split into $\mathbf{w}_C$ and $\mathbf{w}_S$. For the CNN model that is utilized for MNIST and FMNIST, we split the model and allocate the first 4 convolutional layers to the client, and the remaining 1 convolutional layer and 3 FC layers to the server: we have $|\mathbf{w}_C| = 387,840$ and $|\mathbf{w}_S| = 3,480,330$, i.e., $\frac{|\mathbf{w}_C|}{|\mathbf{w}|} = 0.10$. A single FC layer with $23,050$ parameters is utilized as the auxiliary network at the end of $\mathbf{w}_C$. Hence, the size of the auxiliary network is 0.60% of the entire model $\mathbf{w}$. For VGG-11 utilized for CIFAR-10, we split the model as $|\mathbf{w}_C| = 972,554$ and $|\mathbf{w}_S| = 8,258,560$ to have $\frac{|\mathbf{w}_C|}{|\mathbf{w}|} = 0.11$. A FC layer with $10,250$ parameters is utilized as the auxiliary network, which is 0.11% size of the full model $\mathbf{w}$.

Implementation details: At each global round, the server randomly samples $K = 300$ out of $N = 1000$ clients in the system to participate. We consider a fixed learning rate of 0.01 and a momentum of 0.9. The mini-batch size is set to 10, and the number of epochs at each client is set to one: at each global round, each participating client performs 6 local updates for MNIST, FMNIST and 5 local updates for CIFAR-10.

Test accuracy versus communication load: Fig. 2 shows the performance of each method as a function of communication load in an IID setup. The proposed idea performs better than SplitFed since the downlink communication for transmitting the backpropagated signals is not required and the size of the auxiliary network is negligible. FL has the worst performance since the entire model $\mathbf{w}$ is transmitted between the server and the clients at every global round.

Test accuracy versus training time: In Fig. 3, we evaluate the test accuracy of each scheme as a function of training time. The training time is evaluated by the latency results in Table 1, where the parameters are set to $P_C = 1$, $P_S = 100$, $R = 1$, $\beta = 0.2$. Again, it can be seen that our scheme performs better than SplitFed since each client can update the model directly by its local loss function, without waiting for the backpropagated signal from the server. Moreover, FL requires significantly larger communication/computation time compared to our method for transmitting/updating the full model $\mathbf{w}$ at the clients. Table 2 compares the accuracy of each scheme at a specific time (1.5×10^{11} for MNIST, 2.5×10^{11} for FMNIST, and 8×10^{11} for CIFAR-10). The overall results are consistent with the results in Fig. 3, confirming significant advantage of the proposed idea.

Effect of client-side computing power P_C and transmission rate R: In Fig. 4, we observe the performance of each scheme depending on two important parameters, the client-side computing power P_C and the transmission rate R. Other parameters are set to be same as in Fig. 3. If both P_C and R are small, FL requires significant computation/communication time and thus achieves lower performance compared to others. However, as P_C and R increases, FL shows comparable performance with our method since updating the entire model at the clients and transmitting the entire model does not require significant delay with large P_C and R. The overall results confirm the advantage of our scheme in practical scenarios where clients having low computing powers (small P_C) and low transmission rates (small R) aim to train a shared global model.

6. Conclusion

We proposed a new federated split learning algorithm that is both fast and efficient in terms of communication requirements. The key idea is to update the client-side and server-side models in parallel via local-loss-based training highly tailored to split learning. We provided an optimal solution on splitting the model to minimize the latency, and presented a theoretical analysis that guarantees convergence

of the proposed method. Extensive experimental results confirmed the advantage of our idea compared to FL and SplitFed. We believe that our results provide a new direction to the federated/split learning community for training a large-scale model in practical settings.

References

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pp. 583–593. PMLR, 2019.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pp. 736–745. PMLR, 2020.

Gupta, O. and Raskar, R. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016a.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016b.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Laskin, M., Metz, L., Nabarrao, S., Saroufim, M., Noune, B., Luschi, C., Sohl-Dickstein, J., and Abbeel, P. Parallel training of deep networks with local updates. *arXiv preprint arXiv:2012.03837*, 2020.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

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

Nøkland, A. and Eidnes, L. H. Training neural networks with local error signals. In *International Conference on Machine Learning*, pp. 4839–4850. PMLR, 2019.

Thapa, C., Chamikara, M. A. P., and Camtepe, S. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

Vepakomma, P., Gupta, O., Swedish, T., and Raskar, R. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

A. Proof of Theorem 1

Case 1: We first consider the case with

$$\alpha \geq \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}, \tag{9}$$

which is equivalent to $\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C} \geq \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$. Hence, we have $\max\left(\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}\right) = \frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}$. Now the latency of our scheme can be rewritten as

$$\frac{(q|D|+\alpha|\mathbf{w}|)K}{R}+\frac{\alpha\beta|D||\mathbf{w}|}{P_C}+\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \tag{10}$$

which is an increasing function of α. To minimize this latency in the range of $\alpha \geq \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$, the optimal solution is $\alpha = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$.

Case 2: Now consider

$$\alpha \leq \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}, \tag{11}$$

which leads to $\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C} \leq \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$, i.e., $\max\left(\frac{\alpha|\mathbf{w}|K}{R}+\frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}\right) = \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$. In this case, the latency of our scheme becomes

$$\left(\frac{|\mathbf{w}|K}{R}+\frac{\beta|D||\mathbf{w}|}{P_C}-\frac{|D||\mathbf{w}|K}{P_S}\right)\alpha+\frac{q|D|}{R}+\frac{|D||\mathbf{w}|K}{P_S}. \tag{12}$$

Here, if $P_S \leq (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the latency is a decreasing function of α and the optimal solution minimizing the latency becomes $\alpha = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$ in the range of $\alpha \leq \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$. Otherwise, i.e., $P_S > (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, latency is an increasing function of α.

Now we combine the results of both case 1 and case 2. If $P_S \leq (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the optimal solution minimizing the latency becomes $\alpha = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$. Otherwise, i.e., if $P_S > (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the latency is an increasing function of α, which completes the proof.

B. Proof of Theorem 2

For notational simplicity, we let $f_{C,k}(\mathbf{w}_C^t) := F_{C,k}(\mathbf{w}_C^t, \mathbf{a}_C^t)$ and $f_{S,k}(\mathbf{w}_S^t) := F_{S,k}(\mathbf{w}_S^t, \mathbf{w}_C^t)$.

B.1. Convergence of client-side model

Due to the L-smoothness of client-side loss function, we can write

$$F_C(\mathbf{w}_C^{t+1}) \leq F_C(\mathbf{w}_C^t)+\nabla F_C(\mathbf{w}_C^t)^T(\mathbf{w}_C^{t+1}-\mathbf{w}_C^t)+\frac{L}{2}\|\mathbf{w}_C^{t+1}-\mathbf{w}_C^t\|^2. \tag{13}$$

Note that we have

$$\mathbf{w}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t}\left(\mathbf{w}_C^t-\eta_t\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right) \tag{14}$$

$$= \mathbf{w}_C^t-\eta_t\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t) \tag{15}$$

where $\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(x;\mathbf{w}_{C,k})$ for a given mini-batch $\tilde{D}_k \subset D_k$. Hence, we can rewrite (13) as follows:

$$F_C(\mathbf{w}_C^{t+1}) \leq F_C(\mathbf{w}_C^t)-\eta_t\nabla F_C(\mathbf{w}_C^t)^T\left(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right)+\frac{L}{2}\eta_t^2\left\|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right\|^2. \tag{16}$$

Now by taking the expectations at both sides of (16), we have

$$\mathbb{E}[F_C(\mathbf{w}_C^{t+1})] \leq \mathbb{E}[F_C(\mathbf{w}_C^t)] - \eta_t \underbrace{\mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\Big)\Big]}_{B_1} \tag{17}$$

$$+ \frac{L}{2}\eta_t^2 \underbrace{\mathbb{E}\left[\left\|\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\right\|^2\right]}_{B_2} \tag{18}$$

In the following, we will bound B_1 and B_2, respectively.

We first consider B_1. By defining X as

$$X = \frac{1}{K}\sum_{k\in A_t} \Big(\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t) - \nabla f_{C,k}(\mathbf{w}_C^t)\Big), \tag{19}$$

we can find the lower bound of B_1 as

$$B_1 = \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \tag{20}$$

$$= \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(X + \frac{1}{K}\sum_{k\in A_t} \nabla f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \tag{21}$$

$$\geq \underbrace{\mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \nabla f_{C,k}(\mathbf{w}_C^t)\Big)\Big]}_{C_1} - \underbrace{\|\mathbb{E}[\nabla F_C(\mathbf{w}_C^t)^T X]\|}_{C_2}. \tag{22}$$

Since $f_{C,k}(\mathbf{w}_C^t) := F_{C,k}(\mathbf{w}_C^t, \mathbf{a}_C^t)$ and A_t is chosen uniformly at random among N clients in the system, by the law of total expectation, we can write

$$C_1 = \mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{23}$$

Now we consider C_2. Note that since $\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)$ is an unbiased estimator of $\nabla f_{C,k}(\mathbf{w}_C^t)$, we have

$$\|\mathbb{E}[\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t) - \nabla f_{C,k}(\mathbf{w}_C^t)]\| = 0. \tag{24}$$

We also note that $\mathbb{E}[U^T V] \leq \frac{1}{4}\mathbb{E}[\|U\|^2] + \mathbb{E}[\|V\|^2]$ holds for any vectors U and V. Hence, we have

$$C_2 = \big\|\mathbb{E}[\nabla F_C(\mathbf{w}_C^t)^T X]\big\| \tag{25}$$

$$= \big\|\mathbb{E}\left[\mathbb{E}\left[\nabla F_C(\mathbf{w}_C^t)^T X|\Omega\right]\right]\big\| \tag{26}$$

$$= \big\|\mathbb{E}\left[\nabla F_C(\mathbf{w}_C^t)^T \mathbb{E}\left[X|\Omega\right]\right]\big\| \tag{27}$$

$$\leq \frac{1}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2] + \mathbb{E}\left[\|\mathbb{E}\left[X|\Omega\right]\|^2\right] \tag{28}$$

$$= \frac{1}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{29}$$

where the last equality comes from (24).

By inserting (23) and (29) to (22), we obtain

$$B_1 = \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \geq \frac{3}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{30}$$

Now we consider B_2 in (17). We can bound B_2 as

$$B_2 = \|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\|^2 \tag{31}$$

$$\underset{(a)}{\leq} \frac{1}{K}\sum_{k\in A_t}\|\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\|^2 \tag{32}$$

$$= \frac{1}{K}\sum_{k\in A_t}\left\|\frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(x;\mathbf{w}_C^t)\right\|^2 \tag{33}$$

$$\underset{(b)}{\leq} \frac{1}{K}\sum_{k\in A_t}\frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\left\|\nabla\ell(x;\mathbf{w}_C^t)\right\|^2 \tag{34}$$

$$\underset{(c)}{\leq} G_1 \tag{35}$$

where (a) and (b) comes from the Cauchy-Schwarz inequality and (c) comes from Assumption 2.

By inserting (30) and (35) to (17), we obtain

$$\mathbb{E}[F_C(\mathbf{w}_C^{t+1})] \leq \mathbb{E}[F_C(\mathbf{w}_C^t)] - \frac{3}{4}\eta_t\mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] + \frac{G_1 L}{2}\eta_t^2. \tag{36}$$

Now by summing up for all global rounds $t = 0, 1, ...T-1$, we have

$$\mathbb{E}[F_C(\mathbf{w}_C^T)] \leq \mathbb{E}[F_C(\mathbf{w}_C^0)] - \frac{3}{4}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] + \frac{G_1 L}{2}\sum_{t=0}^{T-1}\eta_t^2. \tag{37}$$

Finally from $F_C(\mathbf{w}_C^*) \leq \mathbb{E}[F_C(\mathbf{w}_C^T)]$, we can write

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] \leq \frac{4(F_C(\mathbf{w}_C^0) - F_C(\mathbf{w}_C^*))}{3\Gamma_T} + \frac{G_1 L}{2}\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^2 \tag{38}$$

which completes the proof for the client-side model.

B.2. Convergence of server-side model

Due to the L-smoothness of the server-side loss function, following the same procedure of the client-side model, we have

$$\mathbb{E}[F_S(\mathbf{w}_S^{t+1})] \leq \mathbb{E}[F_S(\mathbf{w}_S^t)] - \eta_t \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\Big)\Big]}_{B_3} \tag{39}$$

$$+ \frac{L}{2}\eta_t^2\underbrace{\mathbb{E}[\|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\|^2]}_{B_4} \tag{40}$$

where $\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(g_{\mathbf{w}_{C,k}^t}(x);\mathbf{w}_S^t)$ for a given mini-batch $\tilde{D}_k \subset D_k$. We will derive the bounds of B_3 and B_4.

We first consider B_3. By defining X as

$$X = \frac{1}{K}\sum_{k\in A_t}\Big(\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) - \nabla f_{S,k}(\mathbf{w}_S^t)\Big), \tag{41}$$

we can write

$$\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\Big)\Big] \tag{42}$$

$$= \mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(X + \frac{1}{K}\sum_{k\in A_t}\nabla f_{S,k}(\mathbf{w}_S^t)\Big)\Big] \tag{43}$$

$$\geq \mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\nabla f_{S,k}(\mathbf{w}_S^t)\Big)\Big] - \|\mathbb{E}[\nabla F_S(\mathbf{w}_S^t)^T X]\| \tag{44}$$

$$\geq \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\nabla F_{S,k}(\mathbf{w}_S^t)\Big)\Big]}_{C_1} + \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big)\Big)\Big]}_{C_2} \tag{45}$$

$$- \underbrace{\|\mathbb{E}[\nabla F_S(\mathbf{w}_S^t)^T X]\|}_{C_3} \tag{46}$$

As can be seen in the derivation for the client-side model, we have

$$C_1 = \mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2] \text{ and } C_3 \leq \frac{1}{4}\mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2]. \tag{47}$$

Now we analyze C_2. We have

$$\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big)\Big) \tag{48}$$

$$\geq -\left\|\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\Big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\Big)\Big)\right\| \tag{49}$$

$$\geq -\big\|\nabla F_S(\mathbf{w}_S^t)\big\|\left\|\frac{1}{K}\sum_{k\in A_t}\big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big)\right\| \tag{50}$$

$$\geq -\sqrt{G_2}\left\|\frac{1}{K}\sum_{k\in A_t}\big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big)\right\| \tag{51}$$

$$\geq -\sqrt{G_2}\frac{1}{K}\sum_{k\in A_t}\big\|\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big\| \tag{52}$$

$$= -\sqrt{G_2}\frac{1}{K}\sum_{k\in A_t}\left\|\frac{1}{|D_k|}\sum_{x\in D_k}\nabla\ell(g_{\mathbf{w}_C^t}(x);\mathbf{w}_S) - \frac{1}{|D_k|}\sum_{x\in D_k}\nabla\ell(g_{\mathbf{w}_C^*}(x);\mathbf{w}_S)\right\| \tag{53}$$

$$= -\sqrt{G_2}\frac{1}{K}\sum_{k\in A_t}\left\|\int\nabla\ell(z;\mathbf{w}_S^t)p_{C,k}^t(z)dz - \int\nabla\ell(z;\mathbf{w}_S^t)p_{C,k}^*(z)dz\right\| \tag{54}$$

$$\geq -\sqrt{G_2}\frac{1}{K}\sum_{k\in A_t}\int\big\|\nabla\ell(z;\mathbf{w}_S^t)\big\|\,\big\|p_{C,k}^t(z) - p_{C,k}^*(z)\big\|\,dz \tag{55}$$

$$\geq -G_2\frac{1}{K}\sum_{k\in A_t}d_{C,k}^t. \tag{56}$$

Now we can write

$$C_2 \geq -\mathbb{E}\left[G_2\frac{1}{K}\sum_{k\in A_t}d_{C,K}^t\right] \tag{57}$$

$$= -G_2\frac{1}{N}\sum_{k=1}^{N}d_{C,k}^t \tag{58}$$

since A_t chosen uniformly at random among N clients in the system.

By utilizing the results of C_1, C_2, C_3, we have

$$B_3 = \mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\Big)\Big] \geq \frac{3}{4}\mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2] - G_2 \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t \tag{59}$$

Following the same procedure of the client-side model, for B_4, we have

$$B_4 = \|\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\|^2 \leq G_2. \tag{60}$$

Now by inserting the results of (59) and (60) to (39), we have

$$\mathbb{E}[F_S(\mathbf{w}_S^{t+1})] \leq \mathbb{E}[F_S(\mathbf{w}_S^t)] - \frac{3}{4}\eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] + \eta_t G_2 \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t + \frac{LG_2}{2}\eta_t^2. \tag{61}$$

Summing up for all global rounds $t = 0, 1, ...T-1$, we have

$$\mathbb{E}[F_S(\mathbf{w}_C^T)] \leq \mathbb{E}[F_S(\mathbf{w}_C^0)] - \frac{3}{4}\sum_{t=0}^{T-1} \eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_C^t)||^2] + G_2 \sum_{t=0}^{T-1}\left(\eta_t \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t + \frac{L}{2}\eta_t^2\right). \tag{62}$$

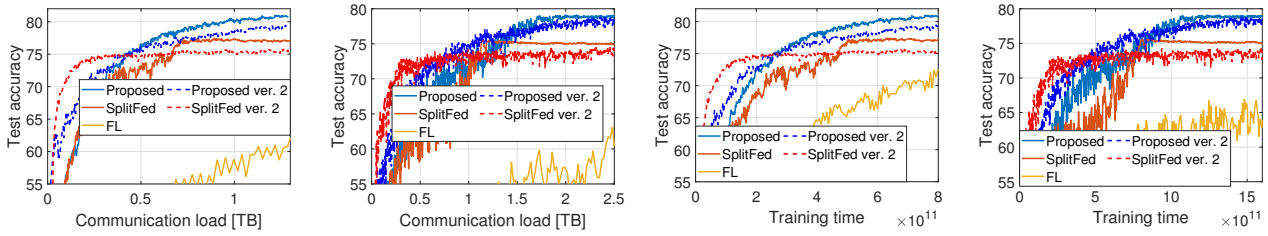
Finally from $F_S(\mathbf{w}_S^*) \leq \mathbb{E}[F_S(\mathbf{w}_S^T)]$, we can write

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1} \eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] \leq \frac{4(F_S(\mathbf{w}_S^0) - F_S(\mathbf{w}_S^*))}{3\Gamma_T} + G_2 \frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t + \frac{L}{2}\eta_t^2\right) \tag{63}$$

which completes the proof for the server-side model.

C. Additional experimental results

Instead of parallelizing the server-side update process of SplitFed and our scheme, one can think of updating the server-side model sequentially in the order of arrivals of the results from the clients (Thapa et al., 2020). Fig. 5 shows the results of this idea where "Proposed ver. 2" and "SplitFed ver. 2" denote schemes that sequential server-side update process is applied to our idea and SplitFed, respectively. The parameters are set to be same as in Fig. 3. As expected, although the sequential update method can speed up training in the beginning, the convergence to the optimal solution is not theoretically guaranteed and thus achieves a lower accuracy.



(a) Comm. load, IID (b) Comm. load, non-IID (c) Training time, IID (d) Training time, non-IID

Figure 5. Effect of sequential update process at the server. CIFAR-10 is utilized for training VGG-11.