Asynchronous Federated Optimization

Cong Xie [1] Oluwasanmi Koyejo [1] Indranil Gupta [1]

Abstract

Federated learning enables training on a massive number of edge devices. To improve flexibility and scalability, we propose a new asynchronous federated optimization algorithm. We prove that the proposed approach has near-linear convergence to a global optimum, for both strongly and non-strongly convex problems, as well as a restricted family of non-convex problems. Empirical results show that the proposed algorithm converges fast and tolerates staleness.

1 Introduction

Federated learning (Konevcnỳ et al., 2016; McMahan et al., 2016) enables training a global model on datasets decentrally located on a massive number of resource-weak edge devices. Federated learning is motivated by the massive data generated in our daily life, by edge devices/IoT such as smart phones, wearable devices, sensors, and in smart homes/buildings. Ideally, the larger amounts of training data from diverse users results in improved representation and generalization of machine-learning models. Federated learning is also motivated by the need for privacy preservation. In some scenarios, on-device training without depositing data in the cloud is legally required by regulations such as US HIPAA laws (HealthInsurance.org, 1996) and Europe's GDPR law (EU, 2018).

Typically, a federated learning system is composed of servers and workers, whose architecture is similar to parameter servers (Li et al., 2014a;b; Ho et al., 2013). The workers train the models locally on the private data on edge devices. The servers aggregate the learned models from the workers, and produce a global model on the cloud/datacenter. To protect the users' privacy, the workers do not expose the training data to the servers, and instead only expose the trained model.

We summarize the key properties of federated learning below:

- **Infrequent task scheduling.** Edge devices typically have weak computational capacity and limited battery time. Unlike the traditional distributed machine learning, on-device federated learning tasks are allowed to be executed only when the device is idle, charging, and connected to unmetered networks (i.e., WiFi) (Bonawitz et al., 2019). The edge devices will ping the servers when they are ready to execute training tasks. The servers will then schedule training tasks on available edge devices. Furthermore, to avoid congesting the network, the server randomizes the check-in time of the workers. As a result, on each edge device, the training task is executed infrequently.
- **Limited communication.** The connection between edge devices and the remote servers may be frequently unavailable, slow, or expensive (in terms of communication costs or in the power of battery). Thus, compared to typical distributed optimization, communication in federated learning needs to be much less frequent.
- **Non-IID training data.** Unlike the traditional distributed machine learning, the data on different devices are not mixed and IID, and thus represent non-identically distributed samples from the population.

We posit that the synchronous flavor of federated optimization is potentially unscalable, inefficient, and inflexible. Previous synchronous training algorithms for federated averaging (McMahan et al., 2016; Bonawitz et al., 2019) can only handle hundreds of devices in parallel, while there are nearly 4 billion mobile phones in total (eMarketer, 2019). Even at smaller scales, like a stadium during a game, there are thousands of devices involved. Too many devices checking in at the same time can congest the network on the server side. Thus, in each global epoch, the server is limited to selecting only from the subset of available devices to trigger the training tasks. Furthermore, since the task scheduling varies from device to device due to limited computational capacity and battery time, it is difficult to synchronize the selected devices at the end of each epoch. Some devices will no longer be available before synchronization. Instead,

[*]Equal contribution [1]Department of Computer Science, University of Illinois Urbana-Champaign, Illinois, USA. Correspondence to: Cong Xie <cx2@illinois.edu>, Oluwasanmi Koyejo <sanmi@illinois.edu>, Indranil Gupta <indy@illinois.edu>.

the server has to determine a timeout threshold to drop the stragglers. If the number of survived devices is too small, the server may have to drop the entire epoch including all the received updates.

To address these issues that arise in synchronous federated optimization, we propose a novel asynchronous federated optimization algorithm. The key idea is to use a weighted average to update the global model. The mixing weight can also be set adaptively as a function of the staleness. We show that taken together, these changes result in an effective asynchronous federated optimization procedure.

The main contributions of our paper are listed as follows:

- We propose a new asynchronous federated optimization algorithm with provable convergence under non-IID settings.
- We show that the proposed approach has near-linear convergence to a global optimum, for both strongly and non-strongly convex problems, as well as a restricted family of non-convex problems.
- We propose strategies for controlling the error caused by asynchrony. We instroduce a mixing hyperparameter which adaptively controls the trade-off between the convergence rate and variance reduction according to the staleness.
- We show empirically that the proposed algorithm converges fast and outperforms synchronous federated optimization.

2 Related Work

Edge computing (Garcia Lopez et al., 2015; Hong et al., 2013) is increasingly applied in various scenarios such as smart home, wearable devices, and sensor networks. Meanwhile, machine-learning applications are also moving from cloud to edge (Cao et al., 2015; Mahdavinejad et al., 2018; Zeydan et al., 2016). Typically, edge devices have weaker computation and communication capacity compared to the workstations and datacenters, due to the weak hardware, limited battery time, and metered networks. As a result, simple machine-learning models such as MobileNet (Howard et al., 2017) have been proposed for the learning tasks on weak devices.

Existing federated optimization methods (Konevcnỳ et al., 2015; 2016; McMahan et al., 2016; Bonawitz et al., 2019) focus on synchronous training. In each global epoch, training tasks are triggered on a subset of workers. However, perhaps due to the bad networking conditions and occasional issues, some worker may fail. When this happens, the server has to wait until a sufficient number of workers respond. Otherwise, the server times out, drops the current epoch, and moves on to the next epoch. As far as we know, this paper is the first to discuss asynchronous training in federated learning with provable convergence.

Asynchronous training (Zinkevich et al., 2009; Lian et al., 2017; Zheng et al., 2017) is widely used in traditional distributed SGD. Typically, asynchronous SGD converges faster than synchronous SGD, especially when the communication latency is high and heterogeneous. However, classic asynchronous SGD directly sends gradients to the servers after each local update, which is not feasible for edge devices due to unreliable and slow communication. In this paper, we take the advantage of asynchronous training, and combine it with federated optimization.

Table 1. Notations and Terminologies.

Notation/Term	**Description**
n	Number of devices
T	Number of global epochs
$[n]$	Set of integers $\{1, \ldots, n\}$
H_{min}	Minimal number of local iterations
H^i_τ	Number of local iterations in the τ^{th} epoch on the ith device
x_t	Global model in the t^{th} epoch on server
$x^i_{\tau,h}$	Model initialized from x_τ, updated in the hth local iteration, on the ith device
$\mathcal{D}^i$	Dataset on the ith device
$z^i_{t,h}$	Data (minibatch) sampled from $\mathcal{D}^i$
γ	Learning rate
α	Mixing hyperparameter
ρ	Regularization weight
$t - \tau$	Staleness
$s(t-\tau)$	Function of staleness for adaptive α
$\lVert \cdot \rVert$	All the norms in this paper are l_2-norms
Device	Where the training data are placed
Worker	One worker on each device, process that trains the model

3 Problem Formulation

We consider federated learning with n devices. On each device, there is a worker process that trains the model on local data. The overall goal is to train a global model $x \in \mathbb{R}^d$ using data from all the devices.

To do so, we consider the following optimization problem:

$$\min_{x \in \mathbb{R}^d} F(x),$$

where $F(x) = \frac{1}{n} \sum_{i \in [n]} \mathbb{E}_{z^i \sim \mathcal{D}^i} f(x; z^i)$, for $\forall i \in [n]$, z^i is sampled from the local data $\mathcal{D}^i$ on the ith device.

Note that different devices have different local datasets, i.e., $\mathcal{D}^i \neq \mathcal{D}^j, \forall i \neq j$. Thus, samples drawn from different devices may have different expectations i.e. in general, $\mathbb{E}_{z^i \sim \mathcal{D}^i} f(x; z^i) \neq \mathbb{E}_{z^j \sim \mathcal{D}^j} f(x; z^j), \forall i \neq j$.

4 METHODOLOGY

A single execution of federated optimization has T global epochs. In the t^{th} epoch, the server receives a locally trained model x_{new} from an arbitrary worker, and updates the global model by weighted averaging:

$$x_t = (1-\alpha)x_{t-1} + \alpha x_{new},$$

where $\alpha \in (0,1)$ is the mixing hyperparameter.

On an arbitrary device i, after receiving a global model x_t (potentially stale) from the server, we locally solve the following regularized optimization problem using SGD for arbitrary number of iterations:

$$\min_{x \in \mathbb{R}^d} \mathbb{E}_{z^i \sim \mathcal{D}^i} f(x; z^i) + \frac{\rho}{2}\|x - x_t\|^2.$$

The server and workers conduct updates asynchronously. The server immediately updates the global model whenever it receives a local model. The communication between the server and workers is non-blocking.

The detailed algorithm is shown in Algorithm 1. The model parameter $x^i_{\tau,h}$ is updated in hth local iteration after receiving x_τ, on the ith device. $z^i_{\tau,h}$ is the data randomly drawn in hth local iteration after receiving x_τ, on the ith device. H^i_τ is the number of local iterations after receiving x_τ, on the ith device. γ is the learning rate and T is the total number of global epochs.

Remark 1. *On the server side, there are two threads running asynchronously in parallel: scheduler and updater. The scheduler periodically triggers training tasks on some workers. The updater receives locally trained models from workers and updates the global model. There could be multiple updater threads with read-write lock on the global model, which improves the throughput. The scheduler randomizes the timing of training tasks to avoid overloading the updater thread, and controls the staleness ($t-\tau$ in the updater thread). We illustrate the overview of the system in*

Intuitively, larger staleness results in greater error when updating the global model. For the local models with large staleness $(t-\tau)$, we decrease α to mitigate the error caused by staleness. As shown in Algorithm 1, optionally, we use a function $s(t-\tau)$ to decide the value of α. We list some choices for $s(t-\tau)$, parameterized by $a > 0, b \geq 0$:

- Linear: $s_a(t-\tau) = \frac{1}{a(t-\tau)+1}$.
- Polynomial: $s_a(t-\tau) = (t-\tau+1)^{-a}$.
- Exponential: $s_a(t-\tau) = \exp(-a(t-\tau))$.
- Hinge: $s_{a,b}(t-\tau) = \begin{cases} 1 & \text{if } t-\tau \leq b \\ \frac{1}{a(t-\tau-b)+1} & \text{otherwise} \end{cases}$.

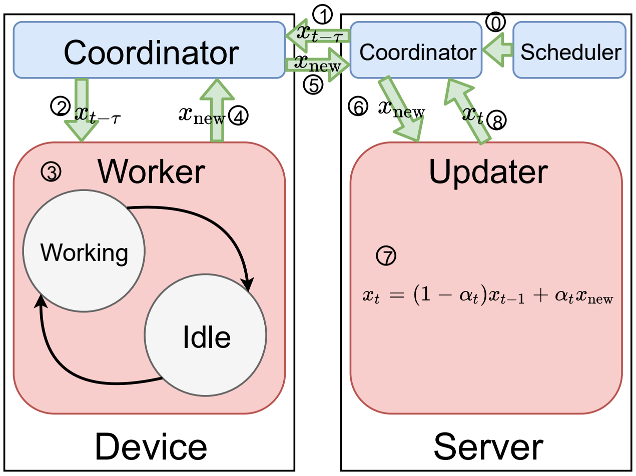


Figure 1. System overview. ⓪: scheduler triggers training tasks through the coordinator. ①, ②: worker receives a delayed global model $x_{t-\tau}$ from server. ③: worker does local update as described in Algorithm 1. The worker process can switch between the two states: working and idle, according to the devices' availability. ④, ⑤, ⑥: worker pushes the locally updated model to server via coordinator. The scheduler queues the models received in ⑤, and feed them to the updater sequentially in ⑥. ⑦, ⑧: server updates the global model and make it ready to read in the coordinator. In our system, ① and ⑤ operates asynchronously in parallel, so that the server can trigger training tasks on devices at any time, and the devices can push the locally updated models to the server at any time.

5 CONVERGENCE ANALYSIS

In this section, we prove the convergence of Algorithm 1 with non-IID data.

5.1 Assumptions

First, we introduce some definitions and assumptions for our convergence analysis.

Definition 1. *(Smoothness) A differentiable function f is L-smooth if for $\forall x, y$,*

$$f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2,$$

where $L > 0$.

Definition 2. *(Strong convexity) A differentiable function f is μ-strongly convex if for $\forall x, y$,*

$$\langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2 \leq f(y) - f(x),$$

where $\mu \geq 0$. Note that if $\mu = 0$, f is convex.

Definition 3. *(Weak convexity) A differentiable function f is μ-weakly convex if the function g with $g(x) = f(x) + \frac{\mu}{2}\|x\|^2$ is convex, where $\mu \geq 0$.*

Algorithm 1 Asynchronous Federated Optimization (FedAsync)

```
Server Process
Input: $\alpha \in (0, 1)$
Initialize $x_0$, $\alpha_t \leftarrow \alpha, \forall t \in [T]$
Scheduler Thread
Scheduler periodically triggers some training tasks on some workers, and sends them the latest global model with time stamp
Updater Thread
for all epoch $t \in [T]$ do
    Receive the pair $(x_{new}, \tau)$ from any worker
    Optional: $\alpha_t \leftarrow \alpha \times s(t - \tau)$, $s(\cdot)$ is a function of the staleness
    $x_t = (1 - \alpha_t)x_{t-1} + \alpha_t x_{new}$
end for
Worker Processes
for all $i \in [n]$ in parallel do
    If triggered by the scheduler:
    Receive the pair of the global model and its time stamp $(x_t, t)$ from the server
    $\tau \leftarrow t$, $x^i_{\tau,0} \leftarrow x_t$
    For $\mu$-weakly convex $F$:
        Define $g_{x_t}(x; z) = f(x; z) + \frac{\rho}{2}\|x - x_t\|^2$, where $\rho > \mu$
    for all local iteration $h \in [H^i_\tau]$ do
        Randomly sample $z^i_{\tau,h} \sim \mathcal{D}^i$
        $x^i_{\tau,h} \leftarrow \begin{cases} \text{Option I, for strongly convex } F\text{:} \\ \quad x^i_{\tau,h-1} - \gamma\nabla f(x^i_{\tau,h-1}; z^i_{\tau,h}) \\ \text{Option II, for weakly convex } F\text{:} \\ \quad x^i_{\tau,h-1} - \gamma\nabla g_{x_t}(x^i_{\tau,h-1}; z^i_{\tau,h}) \end{cases}$
    end for
    Push $(x^i_{\tau,H^i_\tau}, \tau)$ to the server
end for
```

Remark 2. *Note that when f is μ-weakly convex, then f is convex if $\mu = 0$, and potentially non-convex if $\mu > 0$.*

Assumption 1. *(Existence of global optimum) We assume that there exists a set $\mathcal{X}_* \subset \mathbb{R}^d$, where any element $x_* \in \mathcal{X}_*$ is a global optimum of $F(x)$, $x_* = \inf_x F(x)$, and $\nabla F(x_*) = 0$.*

5.2 Convergence Guarantees

Based on the assumptions above, we have the following convergence guarantees. Detailed proofs can be found in the appendix.

Theorem 1. *Assume that the global loss function F is L-smooth and μ-strongly convex, and each worker executes at least H_{min} local updates before pushing models to the server. Furthermore, we assume that for $\forall x \in \mathbb{R}^d, i \in [n]$, and $\forall z \sim \mathcal{D}^i$, we have $\mathbb{E}\|\nabla f(x; z) - \nabla F(x)\|^2 \le V_1$, and $\mathbb{E}\left[\|\nabla f(x; z)\|^2\right] \le V_2$. Taking $\gamma < \frac{1}{L}$, after T global updates/epochs on the server, Algorithm 1 with Option I converges to a global optimum $x_* \in \mathcal{X}_*$:*

$$\begin{aligned}&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\ &\le (\beta)^T \left[F(x_0) - F(x_*)\right] + \left(1 - (\beta)^T\right)\mathcal{O}(V_1 + V_2),\end{aligned}$$

where $\beta = 1 - \alpha + \alpha(1 - \gamma\mu)^{H_{min}}$.

Remark 3. *The mixing hyperparameter $\alpha \in (0, 1)$ controls the trade-off between the convergence rate and additional error caused by variance. When $\alpha \to 1$, the convergence rate approaches $(1 - \gamma\mu)^{TH_{min}}$, with the additional error $\mathcal{O}\left(V_1 + V_2\right)$:*

$$\begin{aligned}&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\ &\le (1 - \gamma\mu)^{TH_{min}}\left[F(x_0) - F(x_*)\right] + \mathcal{O}\left(V_1 + V_2\right).\end{aligned}$$

When $\alpha \to 0$, $\beta \to 1$. As a result, the variance $\left(1 - \beta^T\right)\mathcal{O}\left(V_1 + V_2\right)$ is reduced to 0. In practice, to balance the convergence rate and the variance reduction, we use diminishing α: $\alpha_t \propto \frac{1}{\sqrt{t}}, \forall t \in [T]$, such that the the algorithm converges fast at the beginning, and reduces the variance at the end.

Theorem 2. *Assume that the global loss function F is L-smooth and μ-weakly convex (potentially non-convex), and each worker executes at least H_{min} local updates before pushing models to the server. Furthermore, we assume that for $\forall x \in \mathbb{R}^d, i \in [n]$, and $\forall z \sim \mathcal{D}^i$, we have $\mathbb{E}\|\nabla f(x; z) - \nabla F(x)\|^2 \le V_1$, and $\mathbb{E}\left[\|\nabla g_{x'}(x; z)\|^2\right] \le V_2$, $\forall x'$. Taking $\rho > \mu$ and $\gamma < \min(\frac{1}{L}, \frac{2}{\rho - \mu})$, after T global updates/epochs on the server, Algorithm 1 with Option II converges to a global optimum $x_* \in \mathcal{X}_*$:*

$$\begin{aligned}&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\ &\le (\beta)^T \left[F(x_0) - F(x_*)\right] + \left(1 - (\beta)^T\right)\mathcal{O}(V_1 + V_2),\end{aligned}$$

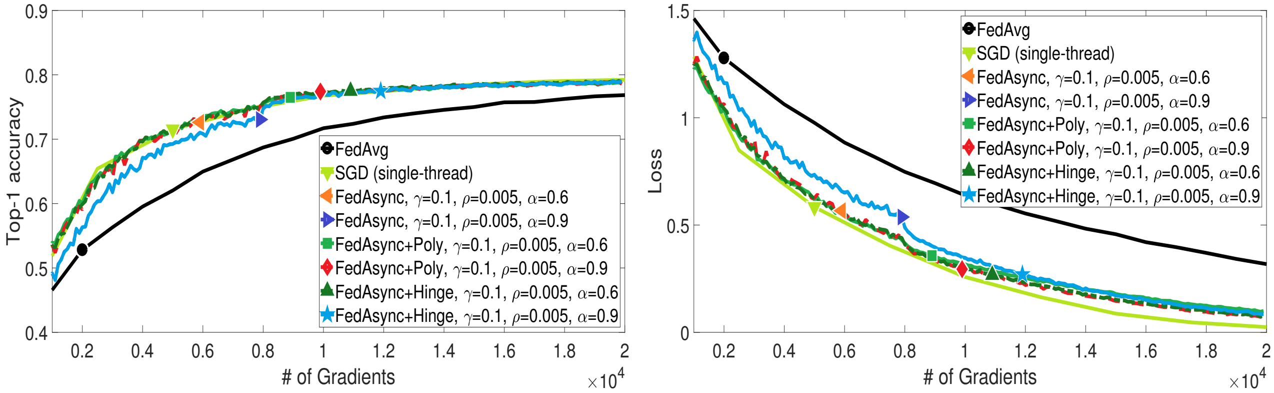
where $\beta = 1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho - \mu)}{2}\right]^{H_{min}}$.

6 Experiments

In this section, we empirically evaluate the proposed algorithm.

6.1 Datasets

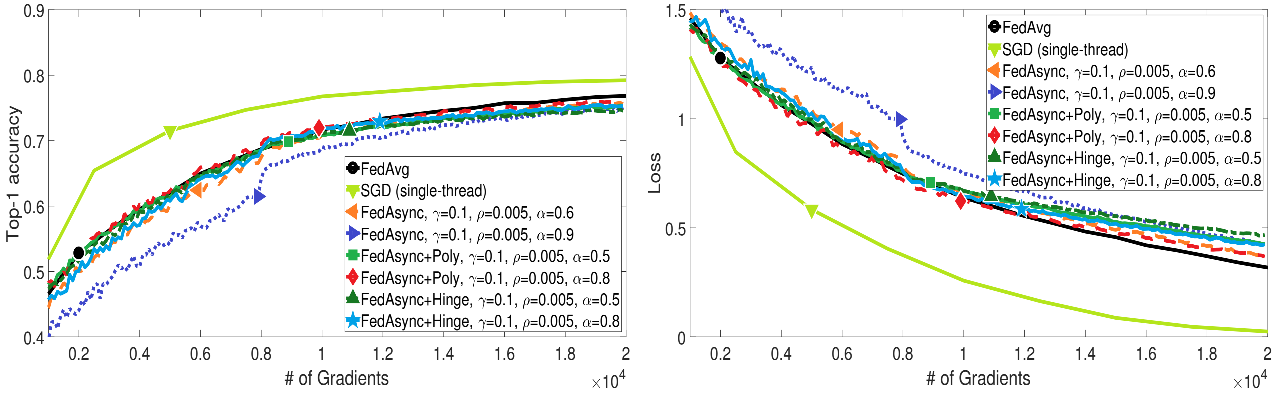
We conduct experiments on the benchmark CIFAR-10 image classification dataset (Krizhevsky & Hinton, 2009), which is composed of 50k images for training and 10k images for testing. We resize each image and crop it to the shape of $(24, 24, 3)$. We use convolutional neural network (CNN) with 4 convolutional layers followed by 1 fully



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 4$

(b) Cross Entropy on Training Set, $t - \tau \leq 4$

Figure 2. Metrics vs. # of gradients. The maximum staleness is 4. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$. Note that when the maximum staleness is 4, *FedAsync* and *FedAsync+Hinge* with $b = 4$ are the same.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 16$

(b) Cross Entropy on Training Set, $t - \tau \leq 16$

Figure 3. Metrics vs. # of gradients. The maximum staleness is 16. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$.

connected layer. We chose a simple network architecture so that it can be easily handled by mobile devices. The detailed network architecture can be found in the appendix. In each experiment, the training set is partitioned onto $n = 100$ devices. Each of the $n = 100$ partitions has 500 images. For any worker, the minibatch size for SGD is 50.

6.2 Evaluation Specifics

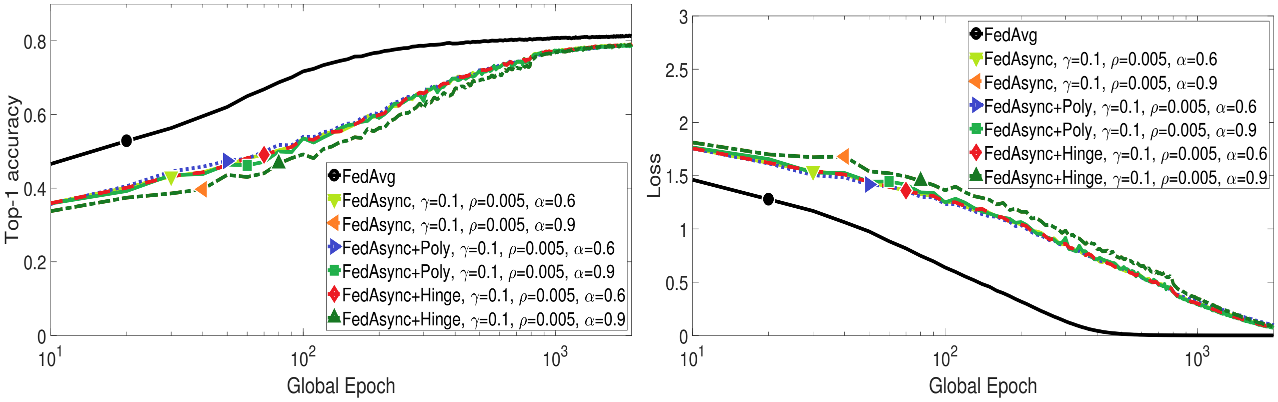
The baseline algorithm is *FedAvg* introduced by McMahan et al. (2016), which is synchronous federated optimization. The detailed *FedAvg* is shown in Algorithm 2. For *FedAvg*, in each epoch, $k = 10$ devices are randomly selected to launch local updates. We also use single-thread SGD as the baseline. The detailed *SGD* is shown in Algorithm 3. For the two baseline algorithms, we use grid search to tune the learning rates and report the best results according to the top-1 accuracy on the testing set.

We repeat each experiment 10 times and take the average. We use top-1 accuracy on the testing set, and cross entropy loss function on the training set as the evaluation metrics.

For convenience, we name Algorithm 1 as *FedAsync*. We also test the performance of *FedAsync* with adaptive mixing hyperparameters $\alpha_t = \alpha \times s(t - \tau)$, as mentioned in Section 4. We employ the following two strategies:

- Polynomial: $s_a(t - \tau) = (t - \tau + 1)^{-a}$.
- Hinge: $s_{a,b}(t - \tau) = \begin{cases} 1 & \text{if } t - \tau \leq b \\ \frac{1}{a(t-\tau-b)+1} & \text{otherwise} \end{cases}$.

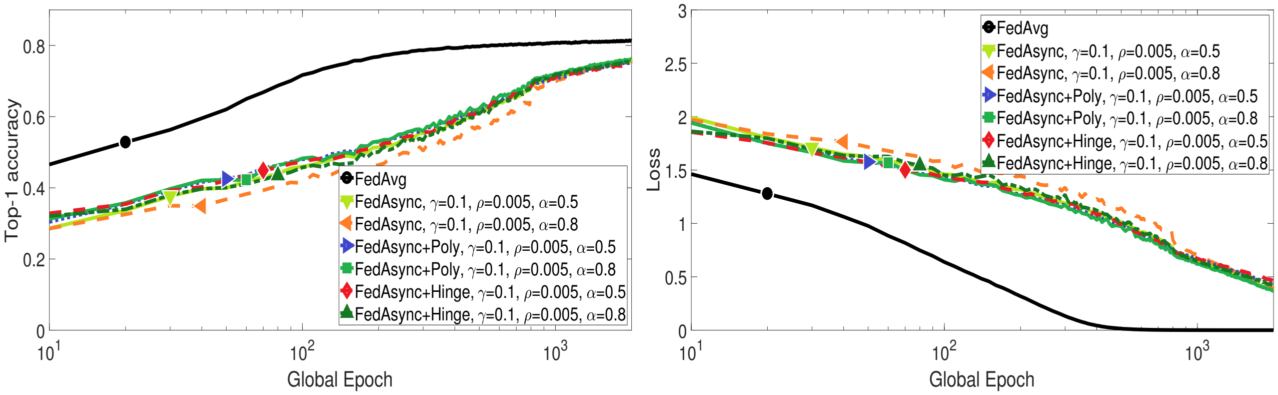
For convenience, we refer to *FedAsync* with polynomial adaptive α as *FedAsync+Poly*, and *FedAsync* with hinge



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 4$

(b) Cross Entropy on Training Set, $t - \tau \leq 4$

Figure 4. Metrics vs. # of global epochs. The maximum staleness is 4. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$. Note that when the maximum staleness is 4, *FedAsync* and *FedAsync+Hinge* with $b = 4$ are the same.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 16$

(b) Cross Entropy on Training Set, $t - \tau \leq 16$

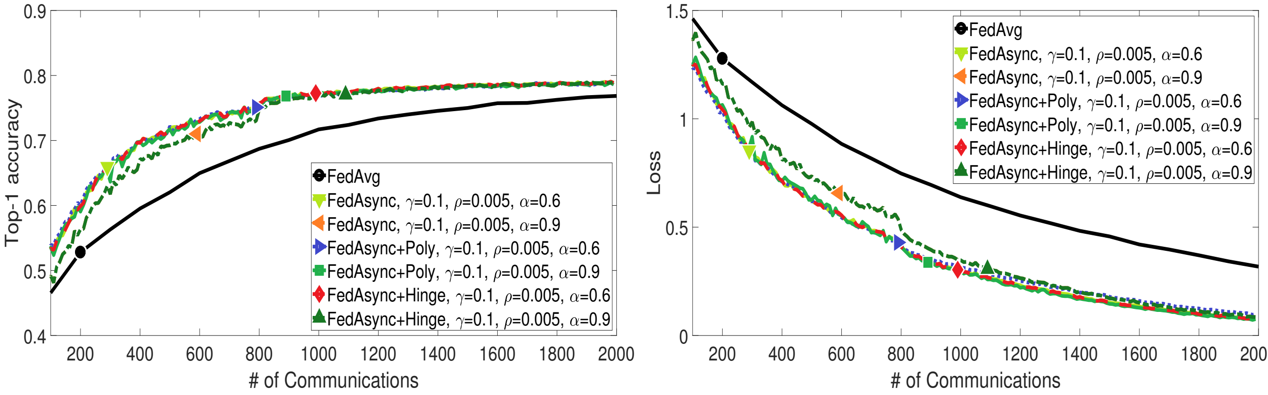
Figure 5. Metrics vs. # of global epochs. The maximum staleness is 16. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$.

adaptive α as *FedAsync+Hinge*.

To compare asynchronous training and synchronous training, we conduct three comparisons: metrics vs. number of global epochs, metrics vs. number of gradients, and metrics vs. number of communications:

- The number of global epochs counts how many times the global model is updated. The total number of global epochs is $T = 2000$ in both Algorithm 1 and Algorithm 2. Single-thread SGD does not have global epochs, so we ignore it in the experiments of metrics vs. # of global epochs.
- The number of gradients is the number of gradients applied to the global model. Note that for both Algorithm 1 and Algorithm 2, an epoch of local iterations is a full pass of the local dataset. Thus, for *FedAsync*, in each global epoch, 10 gradients is applied to the global model. For *FedAvg*, since $k = 10$, $10 \times 10 = 100$ gradients is applied to the global model in each global epoch.
- The number of communications measures the communication overhead on the server side. We count how many times the models are exchanged (sent/received) on the server. On average, in each global epoch, *FedAvg* has $10\times$ the communications of *FedAsync*. Single-thread *SGD* has no communication, so we ignore it.

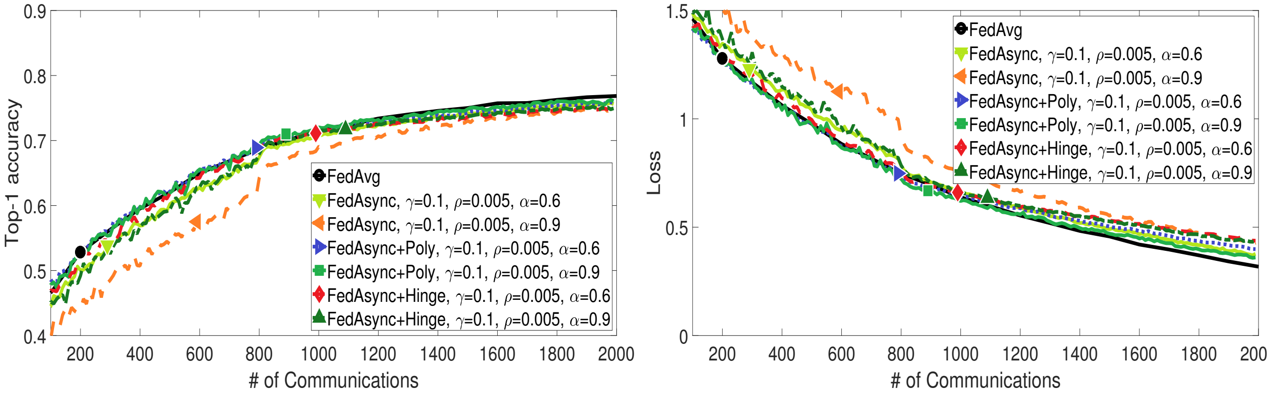
In all the experiments, we simulate the asynchrony by randomly sampling the staleness $(t - \tau)$ from a uniform distribution.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 4$

(b) Cross Entropy on Training Set, $t - \tau \leq 4$

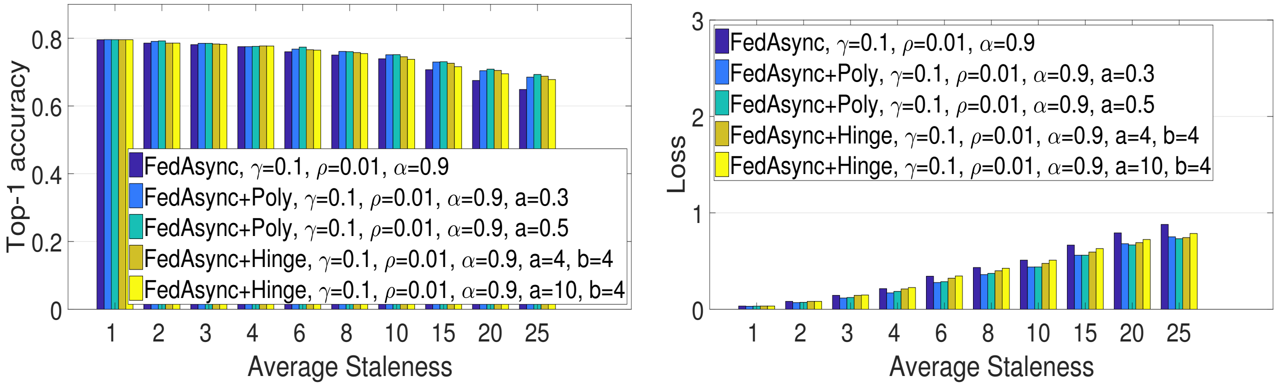
Figure 6. Metrics vs. # of communications. The maximum staleness is 4. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$. Note that when the maximum staleness is 4, *FedAsync* and *FedAsync+Hinge* with $b = 4$ are the same.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 16$

(b) Cross Entropy on Training Set, $t - \tau \leq 16$

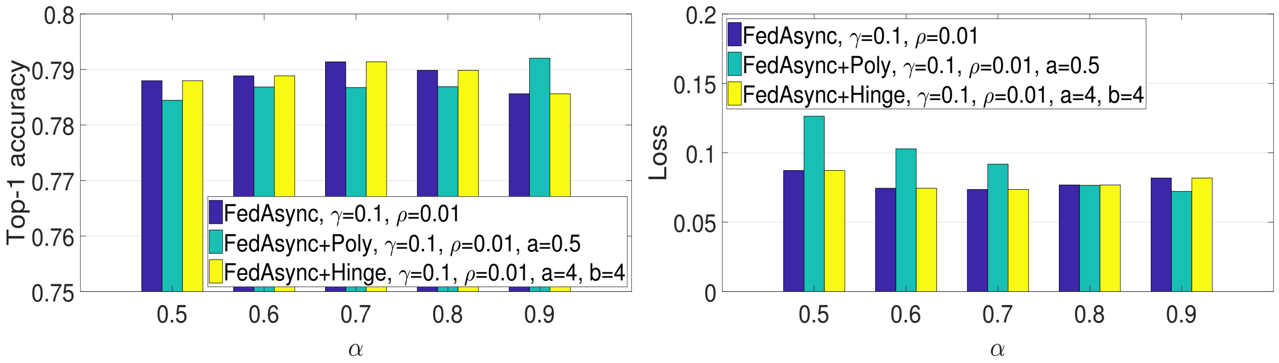
Figure 7. Metrics vs. # of communications. The maximum staleness is 16. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 10, b = 4$.



(a) Top-1 Accuracy on Testing Set

(b) Cross Entropy on Training Set

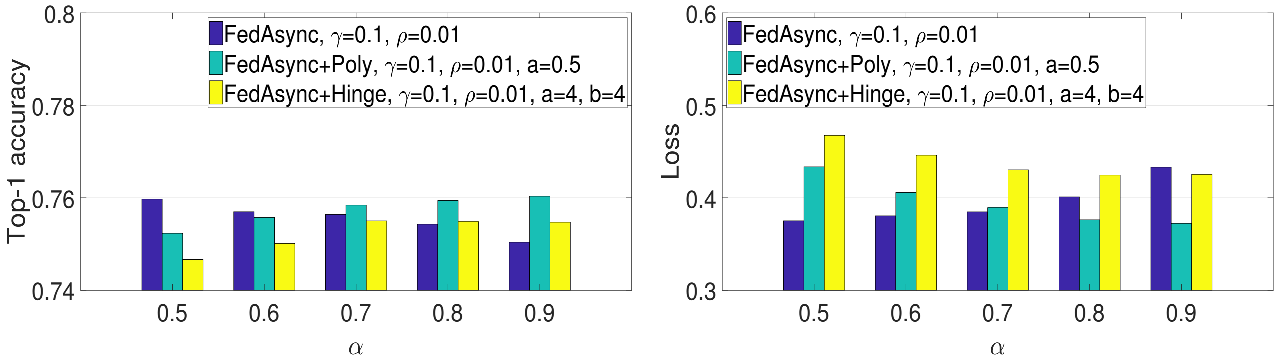
Figure 8. Metrics at the end of training (at the 2000th epoch), with different staleness. α decays by 0.5 at the 800th global epoch.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 4$

(b) Cross Entropy on Training Set, $t - \tau \leq 4$

Figure 9. Metrics at the end of training (at the 2000th epoch), with different α. The maximum staleness is 4. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 4, b = 4$. Note that when the maximum staleness is 4, *FedAsync* and *FedAsync+Hinge* with $b = 4$ are the same.



(a) Top-1 Accuracy on Testing Set, $t - \tau \leq 16$

(b) Cross Entropy on Training Set, $t - \tau \leq 16$

Figure 10. Metrics at the end of training (at the 2000th epoch), with different α. The maximum staleness is 16. α decays by 0.5 at the 800th global epoch. For *FedAsync+Poly*, we take $a = 0.5$. For *FedAsync+Hinge*, we take $a = 4, b = 4$.

6.3 Empirical Results

We test the algorithms with different initial learning rates γ, regularization weights ρ, mixing hyperparameter α, and staleness. α decays by 0.5 at the 800th global epoch.

In Figure 2 and 3, we show how *FedAsync* converges when the number of gradients grows. We can see that when the overall staleness is small, *FedAsync* converges as fast as *SGD*, and faster than *FedAvg*. When the staleness is larger, *FedAsync* converges slower. In the worst case, *FedAsync* has similar convergence rate as *FedAvg*. When α is too large, the convergence can be unstable. Using adaptive α, the convergence can be robust to large α. Note that when the maximum staleness is 4, *FedAsync* and *FedAsync+Hinge* with $b = 4$ are the same.

In Figure 4 and 5, we show how *FedAsync* converges when the number of global epochs grows. Obviously, *FedAvg* makes more progress in each epoch. However, in each epoch, *FedAvg* has to wait until all the $k = 10$ workers respond, while *FedAsync* only needs one worker's response to move on to the next epoch.

In Figure 6 and 7, we show how *FedAsync* converges when the communication overhead grows. With the same amount of communication overhead, *FedAsync* converges faster than *FedAvg* when staleness is small. When staleness is large, *FedAsync* has similar performance as *FedAvg*.

In Figure 8, we show how staleness affects the convergence of *FedAsync*. Overall, larger staleness makes the convergence slower, but the influence is not catastrophic. Furthermore, using adaptive mixing hyperparameters, the instability caused by large staleness can be mitigated.

In Figure 9 and 10, we show how α affects the convergence of *FedAsync*. In general, *FedAsync* is robust to different α. Note that the difference is so tiny that we have to zoom in. When the staleness is small, adaptive mixing hyper-

Algorithm 2 Federated Averaging (FedAvg)

```
Input: $k \in [n]$
Initialize $x_0$
for all epoch $t \in [T]$ do
    Randomly select a group of $k$ workers, denoted as $S_t \subseteq [n]$
    for all $i \in S_t$ in parallel do
        Receive the latest global model $x_{t-1}$ from the server
        $x^i_{t,0} \leftarrow x_{t-1}$
        for all local iteration $h \in [H^i_t]$ do
            Randomly sample $z^i_{t,h}$
            $x^i_{t,h} \leftarrow x^i_{t,h-1} - \gamma \nabla f(x^i_{t,h-1}; z^i_{t,h})$
        end for
        Push $x^i_{t,H^i_t}$ to the server
    end for
    Update the global model: $x_t = \frac{1}{k} \sum_{i \in S_t} x^i_{t,H^i_t}$
end for
```

Algorithm 3 SGD (Single Thread)

```
Initialize $x_0$
for all iteration $t \in [T]$ do
    Randomly sample $z_t$
    $x_t \leftarrow x_{t-1} - \gamma \nabla f(x_{t-1}; z_t)$
end for
```

parameter is less necessary. When the staleness is large, smaller α is better for *FedAsync*, while larger α is better for *FedAsync+Poly* and *FedAsync+Hinge*. That is because adaptive α is automatically adjusted to be smaller when the staleness is large, so that we should not manually decrease α.

6.4 Discussion

In general, the convergence rate of *FedAsync* is between single-thread *SGD* and *FedAvg*. Larger α and smaller staleness makes *FedAsync* closer to single-thread *SGD*. Smaller α and larger staleness makes *FedAsync* closer to *FedAvg*.

FedAsync is generally insensitive to hyperparameters. When the staleness is large, we can tune α to improve the convergence. Without adaptive α, smaller α is better for larger staleness. For adaptive α, the best choice is *FedAsync+Poly* with $s_a(t-\tau) = (t-\tau+1)^{-a}, a = 0.5$.

Larger staleness makes the convergence slower and unstable. There are three ways to control the influence of staleness:

- On the serve side, the updater thread can drop the updates with large staleness $(t-\tau)$. This can also be viewed as a special case of adaptive mixing hyperparameter α. In particular, when the staleness is too large, we can simply take $\alpha = 0$.
- More generally, using adaptive mixing hyperparameters improves the convergence, as shown in Figure 8. Different strategies has different improvement. So far we find that *FedAsync+Poly* with $s_a(t-\tau) = (t-\tau+1)^{-a}, a = 0.5$ has the best performance.
- On the server side, the scheduler thread can control the assignment of training tasks to the workers. If the on-device training is triggered less frequently, the overall staleness will be smaller.

Systematically, *FedAsync* has the following advantages compared to *FedAvg*:

- **Efficiency**: The server can receive the updates from the workers at any time. Unlike *FedAvg*, stragglers' updates will not be dropped. When the staleness is small, *FedAsync* converges much faster than *FedAvg*. In the worst case, when the staleness is large, *FedAsync* still has similar performance as *FedAvg*.
- **Flexibility**: If some workers are no longer eligible for the training tasks (the devices are no longer idle, charging, or connected to unmetered networks), they can temporarily save the workspace, and continue the training or push the trained model to the server later. This also gives more flexibility to the scheduler on the server. Unlike *FedAvg*, *FedAsync* can schedule training tasks even if the workers are currently ineligible, since the server does not wait until the workers respond. The currently ineligible workers can start the training tasks later.
- **Scalability**: Compared to *FedAvg*, *FedAsync* can handle more workers running in parallel since all the updates on the server and the workers are non-blocking. The server only needs to randomize the responding time of the workers to avoid congesting the network.

7 Conclusion

We proposed a novel asynchronous federated optimization algorithm on non-IID training data. The algorithm has near-linear convergence to a global optimum, for both strongly and non-strongly convex problems, as well as a restricted family of non-convex problems. For future work, we plan to investigate the design of strategies to adaptively tune the mixing hyperparameters.

References

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecny, J., Mazzocchi, S., McMahan, H. B., et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Cao, Y., Hou, P., Brown, D., Wang, J., and Chen, S. Distributed analytics and edge intelligence: Pervasive health monitoring at the era of fog computing. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pp. 43–48. ACM, 2015.

eMarketer. Number of mobile phone users worldwide from 2015 to 2020 (in billions). 2019. https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users\-worldwide/, Last visited: Mar. 2019.

EU. European Union's General Data Protection Regulation (GDPR). 2018. https://eugdpr.org/, Last visited: Nov. 2018.

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., and Riviere, E. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

HealthInsurance.org, S. A. Health insurance portability and accountability act of 1996. *Public law*, 104:191, 1996.

Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., Gibson, G. A., Ganger, G., and Xing, E. P. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pp. 1223–1231, 2013.

Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., and Koldehofe, B. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pp. 15–20. ACM, 2013.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Konevcnỳ, J., McMahan, B., and Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

Konevcnỳ, 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*, 2016.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pp. 583–598, 2014a.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pp. 19–27, 2014b.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

Mahdavinejad, M. S., Rezvan, M., Barekatain, M., Adibi, P., Barnaghi, P., and Sheth, A. P. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Zeydan, E., Bastug, E., Bennis, M., Kader, M. A., Karatepe, I. A., Er, A. S., and Debbah, M. Big data caching for networking: Moving from cloud to edge. *IEEE Communications Magazine*, 54(9):36–42, 2016.

Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.-M., and Liu, T.-Y. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4120–4129. JMLR. org, 2017.

Zinkevich, M., Langford, J., and Smola, A. J. Slow learners are fast. In *Advances in neural information processing systems*, pp. 2331–2339, 2009.

Appendix

8 PROOFS

Theorem 1. *Assume that the global loss function F is L-smooth and μ-strongly convex, and each worker execute at least H_{min} local updates before pushing models to the server. Furthermore, we assume that for $\forall x \in \mathbb{R}^d$, $i \in [n]$, and $\forall z \sim \mathcal{D}^i$, we have $\mathbb{E}\|\nabla f(x;z) - \nabla F(x)\|^2 \leq V_1$, and $\mathbb{E}\left[\|\nabla f(x;z)\|^2\right] \leq V_2$. Taking $\gamma < \frac{1}{L}$, after T global updates on the server, Algorithm 1 with Option I converges to a global optimum $x_* \in \mathcal{X}_*$:*

$$\begin{aligned}&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\ &\leq \beta^T \left[F(x_0) - F(x_*)\right] + \left(1 - \beta^T\right) \mathcal{O}(V_1 + V_2),\end{aligned}$$

where $\beta = 1 - \alpha + \alpha(1 - \gamma\mu)^{H_{min}}$.

Proof. Without loss of generality, we assume that in the t^{th} epoch, the server receive the model x_{new}, with time stamp τ. We assume that x_{new} is the result of applying $H \geq H_{min}$ local updates to x_τ on the ith device. We also ignore i in $x^i_{\tau,h}$ and $z^i_{\tau,h}$ for convenience.

Thus, using smoothness and strong convexity, conditional on $x_{\tau,h-1}$, for $\forall h \in [H]$ we have

$$\begin{aligned}&\mathbb{E}\left[F(x_{\tau,h}) - F(x_*)\right] \\ &\leq F(x_{\tau,h-1}) - F(x_*) - \gamma\mathbb{E}\left[\langle \nabla F(x_{\tau,h-1}), \nabla f(x_{\tau,h-1}; z_{\tau,h})\rangle\right] + \frac{L\gamma^2}{2}\mathbb{E}\left[\|\nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] \\ &\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \frac{\gamma}{2}\mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] \\ &\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \frac{\gamma V_1}{2} \\ &\leq F(x_{\tau,h-1}) - F(x_*) - \gamma\mu\left[F(x_{\tau,h-1}) - F(x_*)\right] + \frac{\gamma V_1}{2} \qquad \triangleright\, F(x) \leq F(x_*) + \tfrac{1}{2\mu}\|\nabla F(x)\|^2, \forall x \\ &\leq (1 - \gamma\mu)\left[F(x_{\tau,h-1}) - F(x_*)\right] + \frac{\gamma V_1}{2}.\end{aligned}$$

By telescoping and taking total expectation, after H local updates, we have

$$\begin{aligned}&\mathbb{E}\left[F(x_{\tau,H}) - F(x_*)\right] \\ &\leq (1 - \gamma\mu)^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{\gamma V_1}{2}\sum_{h=1}^{H}(1 - \gamma\mu)^{h-1} \\ &\leq (1 - \gamma\mu)^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{\gamma V_1}{2}\sum_{h=1}^{H_{max}}(1 - \gamma\mu)^{h-1} \\ &\leq (1 - \gamma\mu)^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{\gamma V_1}{2}\frac{1 - (1 - \gamma\mu)^{H_{max}}}{1 - (1 - \gamma\mu)} \\ &\leq (1 - \gamma\mu)^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{\gamma V_1}{2}\frac{H_{max}\gamma\mu}{1 - (1 - \gamma\mu)} \qquad \triangleright\, \gamma\mu \leq 1, 1 - (1 - \gamma\mu)^{H_{max}} \leq H_{max}\gamma\mu \\ &\leq (1 - \gamma\mu)^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{H_{max}\gamma V_1}{2} \\ &\leq (1 - \gamma\mu)^{H_{min}} \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{H_{max}\gamma V_1}{2}.\end{aligned}$$

On the server side, we have $x_t = (1-\alpha)x_{t-1} + \alpha x_{\tau,H}$. Thus, conditional on all $x_{t'}, \forall t' < t$, we have

$$
\begin{aligned}
&\mathbb{E}\left[F(x_t) - F(x_*)\right] \\
&\leq (1-\alpha)\left[F(x_{t-1}) - F(x_*)\right] + \alpha \mathbb{E}\left[F(x_{\tau,H}) - F(x_*)\right] && \triangleright \text{ convexity} \\
&\leq (1-\alpha)\left[F(x_{t-1}) - F(x_*)\right] + \alpha \left[(1-\gamma\mu)^{H_{min}}\left[F(x_\tau) - F(x_*)\right] + \frac{H_{max}\gamma V_1}{2}\right] && \triangleright\, x_{\tau,0} = x_\tau \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{\alpha H_{max}\gamma V_1}{2} + \alpha(1-\gamma\mu)^{H_{min}}\left[F(x_\tau) - F(x_{t-1})\right] \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{\alpha H_{max}\gamma V_1}{2} + \alpha(1-\gamma\mu)^{H_{min}}\left[F(x_\tau) - F(x_*)\right] \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{\alpha H_{max}\gamma V_1}{2} + \alpha(1-\gamma\mu)^{H_{min}}\frac{1}{2\mu}\|\nabla F(x_\tau)\|^2 \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{\alpha H_{max}\gamma V_1}{2} + \frac{\alpha}{2\mu}(1-\gamma\mu)^{H_{min}}V_2 \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{\alpha(V_1+V_2)}{2\mu}.
\end{aligned}
$$

By telescoping and taking total expectation, after T global updates on the server, we have

$$
\begin{aligned}
&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\
&\leq \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)^T\left[F(x_0) - F(x_*)\right] + \frac{V_1+V_2}{2\gamma\mu^2}\left[1 - \left(1-\alpha+\alpha(1-\gamma\mu)^{H_{min}}\right)^T\right].
\end{aligned}
$$

□

Theorem 2. *Assume that the global loss function F is L-smooth and μ-weakly convex (potentially non-convex), and each worker execute at least H_{min} local updates before pushing models to the server. Furthermore, we assume that for $\forall x \in \mathbb{R}^d, i \in [n]$, and $\forall z \sim \mathcal{D}^i$, we have $\mathbb{E}\|\nabla f(x;z) - \nabla F(x)\|^2 \leq V_1$, and $\mathbb{E}\left[\|\nabla g_{x'}(x;z)\|^2\right] \leq V_2$, $\forall x'$. Taking $\rho > \mu$ and $\gamma < \min(\frac{1}{L}, \frac{2}{\rho-\mu})$, after T global updates on the server, Algorithm 1 with Option II converges to a global optimum $x_* \in \mathcal{X}_*$:*

$$
\begin{aligned}
&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\
&\leq \beta^T\left[F(x_0) - F(x_*)\right] + \left(1-\beta^T\right)\mathcal{O}(V_1+V_2),
\end{aligned}
$$

where $\beta = 1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}$.

Proof. Without loss of generality, we assume that in the t^{th} epoch, the server receive the model x_{new}, with time stamp τ. We assume that x_{new} is the result of applying $H \geq H_{min}$ local updates to x_τ on the ith device. We also ignore i in $x^i_{\tau,h}$ and $z^i_{\tau,h}$ for convenience.

Thus, using smoothness and strong convexity, conditional on $x_{\tau,h-1}$, for $\forall h \in [H]$ we have

$$\begin{aligned}
&\mathbb{E}\left[F(x_{\tau,h}) - F(x_*)\right] \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \gamma \mathbb{E}\left[\langle \nabla F(x_{\tau,h-1}), \nabla g_{x_\tau}(x_{\tau,h-1}; z_{\tau,h})\rangle\right] + \frac{L\gamma^2}{2}\mathbb{E}\left[\|\nabla g_{x_\tau}(x_{\tau,h-1}; z_{\tau,h})\|^2\right] \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \frac{\gamma}{2}\mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla g_{x_\tau}(x_{\tau,h-1}; z_{\tau,h})\|^2\right] \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \gamma \mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] + \gamma\rho^2\|x_{\tau,h-1} - x_\tau\|^2 \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \gamma \mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] \\
&\quad + \frac{\gamma\rho^2\|\nabla G_{x_\tau}(x_{\tau,h-1}) - \nabla G_{x_\tau}(x_\tau)\|^2}{(\rho-\mu)^2} \qquad \triangleright\, G_{x_\tau} = F(x) + \tfrac{\rho}{2}\|x - x_\tau\|^2 \text{ is } (\rho-\mu)\text{-strongly convex} \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \gamma \mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] + \frac{4\gamma\rho^2 V_2}{(\rho-\mu)^2}.
\end{aligned}$$

Note that for $\forall x$, we have $(\rho-\mu)$-strongly convex function $G_{x_*}(x) = F(x) + \frac{\rho}{2}\|x - x_*\|^2$. Thus, we have

$$F(x) - F(x_*) \leq G_{x_*}(x) - G_{x_*}(x_*) \leq \frac{\|\nabla G_{x_*}(x)\|^2}{2(\rho-\mu)} \leq \frac{\|\nabla F(x)\|^2 + \rho^2\|x - x_*\|^2}{\rho-\mu} \leq \frac{\|\nabla F(x)\|^2}{\rho-\mu} + \frac{4\rho^2 V_2}{(\rho-\mu)^3}.$$

Thus, we have

$$\begin{aligned}
&\mathbb{E}\left[F(x_{\tau,h}) - F(x_*)\right] \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma}{2}\|\nabla F(x_{\tau,h-1})\|^2 + \gamma \mathbb{E}\left[\|\nabla F(x_{\tau,h-1}) - \nabla f(x_{\tau,h-1}; z_{\tau,h})\|^2\right] + \frac{4\gamma\rho^2 V_2}{(\rho-\mu)^2} \\
&\leq F(x_{\tau,h-1}) - F(x_*) - \frac{\gamma(\rho-\mu)}{2}\left[F(x_{\tau,h-1}) - F(x_*)\right] + \frac{2\gamma\rho^2 V_2}{(\rho-\mu)^2} + \gamma V_1 + \frac{4\gamma\rho^2 V_2}{(\rho-\mu)^2} \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]\left[F(x_{\tau,h-1}) - F(x_*)\right] + \gamma V_1 + \frac{6\gamma\rho^2 V_2}{(\rho-\mu)^2} \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]\left[F(x_{\tau,h-1}) - F(x_*)\right] + C_1. \qquad \triangleright\, C_1 = \gamma V_1 + \tfrac{6\gamma\rho^2 V_2}{(\rho-\mu)^2}
\end{aligned}$$

By telescoping and taking total expectation, after H local updates, we have

$$\begin{aligned}
&\mathbb{E}\left[F(x_{\tau,H}) - F(x_*)\right] \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^H \left[F(x_{\tau,0}) - F(x_*)\right] + C_1 \sum_{h=1}^{H}\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{h-1} \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^H \left[F(x_{\tau,0}) - F(x_*)\right] + C_1 \sum_{h=1}^{+\infty}\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{h-1} \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^H \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{2C_1}{\gamma(\rho-\mu)} \\
&\leq \left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}} \left[F(x_{\tau,0}) - F(x_*)\right] + \frac{2C_1}{\gamma(\rho-\mu)}.
\end{aligned}$$

On the server side, we have $x_t = (1-\alpha)x_{t-1} + \alpha x_{\tau,H}$. Thus, conditional on all $x_{t'}, \forall t' < t$, we have

$$
\begin{aligned}
&\mathbb{E}\left[F(x_t) - F(x_*)\right] \\
&\leq \mathbb{E}\left[G_{x_{t-1}}(x_t) - F(x_*)\right] \\
&\leq (1-\alpha)\left[G_{x_{t-1}}(x_{t-1}) - F(x_*)\right] + \alpha\mathbb{E}\left[G_{x_{t-1}}(x_{\tau,H}) - F(x_*)\right] && \triangleright \text{ convexity of } G_{x_{t-1}}(x) \\
&\leq (1-\alpha)\left[F(x_{t-1}) - F(x_*)\right] + \alpha\mathbb{E}\left[F(x_{\tau,H}) - F(x_*)\right] + \frac{\alpha\rho}{2}\mathbb{E}\left[\|x_{\tau,H} - x_{t-1}\|^2\right] \\
&\leq (1-\alpha)\left[F(x_{t-1}) - F(x_*)\right] + \alpha\mathbb{E}\left[F(x_{\tau,H}) - F(x_*)\right] + \frac{2\alpha\rho V_2}{(\rho-\mu)^2} \\
&\leq (1-\alpha)\left[F(x_{t-1}) - F(x_*)\right] + \alpha\left[\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\left[F(x_\tau) - F(x_*)\right] + \frac{2C_1}{\gamma(\rho-\mu)}\right] + \frac{2\alpha\rho V_2}{(\rho-\mu)^2} \\
&\leq \left(1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{2\alpha C_1}{\gamma(\rho-\mu)} + \frac{2\alpha\rho V_2}{(\rho-\mu)^2} \\
&\quad + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\left[F(x_\tau) - F(x_{t-1})\right] \\
&\leq \left(1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{2\alpha C_1}{\gamma(\rho-\mu)} + \frac{2\alpha\rho V_2}{(\rho-\mu)^2} \\
&\quad + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\left[F(x_\tau) - F(x_*)\right] \\
&\leq \left(1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \frac{2\alpha C_1}{\gamma(\rho-\mu)} + \frac{2\alpha\rho V_2}{(\rho-\mu)^2} + \frac{\alpha V_2}{2(\rho-\mu)} \\
&\leq \left(1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right)\left[F(x_{t-1}) - F(x_*)\right] + \alpha C_2. && \triangleright\, C_2 = \tfrac{2C_1}{\gamma(\rho-\mu)} + \tfrac{2\rho V_2}{(\rho-\mu)^2} + \tfrac{V_2}{2(\rho-\mu)}
\end{aligned}
$$

By telescoping and taking total expectation, after T global updates on the server, we have

$$
\begin{aligned}
&\mathbb{E}\left[F(x_T) - F(x_*)\right] \\
&\leq \left[1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right]^T\left[F(x_0) - F(x_*)\right] + \frac{2C_2}{\gamma(\rho-\mu)}\left[1 - \left[1 - \alpha + \alpha\left[1 - \frac{\gamma(\rho-\mu)}{2}\right]^{H_{min}}\right]^T\right].
\end{aligned}
$$

□

9 EXPERIMENT DETAILS

In Table 2, we show the detailed network structures of the CNN used in our experiments.

A PLEASE ADD SUPPLEMENTAL MATERIAL AS APPENDIX HERE

Put anything that you might normally include after the references as an appendix here, *not in a separate supplementary file*. Upload your final camera-ready as a single pdf, including all appendices.

Table 2. CNN Summary

Layer (type)	Parameters	Input Layer
conv1(Convolution)	channels=64, kernel_size=3, padding=1	data
activation1(Activation)	null	conv1
batchnorm1(BatchNorm)	null	activation1
conv2(Convolution)	channels=64, kernel_size=3, padding=1	batchnorm1
activation2(Activation)	null	conv2
batchnorm2(BatchNorm)	null	activation2
pooling1(MaxPooling)	pool_size=2	batchnorm2
dropout1(Dropout)	probability=0.25	pooling1
conv3(Convolution)	channels=128, kernel_size=3, padding=1	dropout1
activation3(Activation)	null	conv3
batchnorm3(BatchNorm)	null	activation3
conv4(Convolution)	channels=128, kernel_size=3, padding=1	batchnorm3
activation4(Activation)	null	conv4
batchnorm4(BatchNorm)	null	activation4
pooling2(MaxPooling)	pool_size=2	batchnorm4
dropout2(Dropout)	probability=0.25	pooling2
flatten1(Flatten)	null	dropout2
fc1(FullyConnected)	#output=512	flatten1
activation5(Activation)	null	fc1
dropout3(Dropout)	probability=0.25	activation5
fc3(FullyConnected)	#output=10	dropout3
softmax(SoftmaxOutput)	null	fc3