An Optimal Stopping Approach for Iterative Training in Federated Learning

Pengfei Jiang
ECEE
Arizona State University
Tempe, US
pjiang9@asu.edu

Lei Ying
EECS
University of Michigan
Ann Arbor, US
leiying@umich.edu

2020 54th Annual Conference on Information Sciences and Systems (CISS) 978-1-7281-4085-8/20/$31.00 ©2020 IEEE 10.1109/CISS48834.2020.1570616094

***Abstract*—This paper studies the problem of iterative training in Federated Learning. We consider a system with a single parameter server (PS) and M client devices for training a predictive learning model with distributed data sets on the client devices. The clients communicate with the parameter server using a common wireless channel, so each time only one device can transmit. The training is an iterative process consisting of multiple rounds. At beginning of each round (also called an iteration), each client trains the model, broadcast by the parameter server at the beginning of the round, with its own data. After finishing training, the device transmits the update to the parameter server when the wireless channel is available. The server aggregates updates to obtain a new model and broadcasts it to all clients to start a new round. We consider adaptive training where the parameter server decides when to stop/restart a new round, and formulate the problem as an optimal stopping problem. While this optimal stopping problem is difficult to solve, we propose a modified optimal stopping problem. We first develop a low complexity algorithm to solve the modified problem, which also works for the original problem. Experiments on a real data set shows significant improvements compared with policies collecting a fixed number of updates in each round.**

***Index Terms*—Distributed Machine Learning, Federated Learning, Optimal Stopping**

I. Introduction

Most existing machine learning applications for big-data analytics require the models to be trained in data centers, which raises significant privacy concerns when data used contain sensitive personal information such as clicks, photos, etc. Federated learning is a distributed machine learning framework proposed by Google[1] to train a machine learning model with datasets distributed over local devices (such as mobile phones) instead of in data centers. Training process is run on distributed device such as mobile phones so that a device does not need to expose personal data on the device to servers or other devices. The updates for the model (e.g. the gradients of SGD) will be transmitted to a parameter server which will aggregate the updates to update the machine learning model. The updated model will then be broadcast to the devices for the next iteration of training.

Federated learning has applications in many areas [1], e.g. Google has implemented federated learning in their Gboard [2], [3], where a neural network language model is trained using data on personal mobile devices for next-word prediction. Due to randomness and uncertain in data processing and transmissions, it has been observed [4], [5] that even with dedicated servers, learning can be slowed down significantly by a few machines that take unusually long time to complete the training. The problem becomes even worse in Federated Learning where devices have heterogeneous capacities, and are less reliable. Therefore, a critical problem in Federated Learning is to schedule the training, in particular for those machine learning models that require iterative training. In the past, [6], [7] have studied the convergence of the loss function with respect to the number of local iterations on each client and proposed mechanisms to optimally select the number of local iterations on each client. This paper considers a different problem that when the parameter server should stop the current round, update the machine learning model, and start the next round. Such a decision is based on the number of updates received, the expected waiting time to receive the next update, and how the loss function decreases as the number of updates increases. We formulate the problem as an optimal stopping problem and develop a low complexity algorithm to solve the stopping rule. Our experiments on real datasets show significant improvements on training time compared with the policies that collect a fixed number of updates at each round.

[1]https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

II. Problem Formulation

We consider a system with a single parameter server and M client devices such as mobile phones, where each client owns a local dataset. The system is used to train a learning model using the local datasets in an iterative fashion. Each iteration is called a "round". At the beginning of each round, the parameter server broadcasts the latest parameters (such as the parameters of the neural network) to the clients. After receiving the parameters, each client trains the model using its local dataset, e.g. calculate the gradients using SGD, and then transmits the updates (e.g. the gradients) to the parameter server, which aggregates the updates to obtain a new model. This finishes one round, and the next round starts when the parameter server broadcasts the new parameters to the clients. We further assume following idealized data processing and communication models.

978-1-7281-4085-8/20/$31.00 ©2020 IEEE

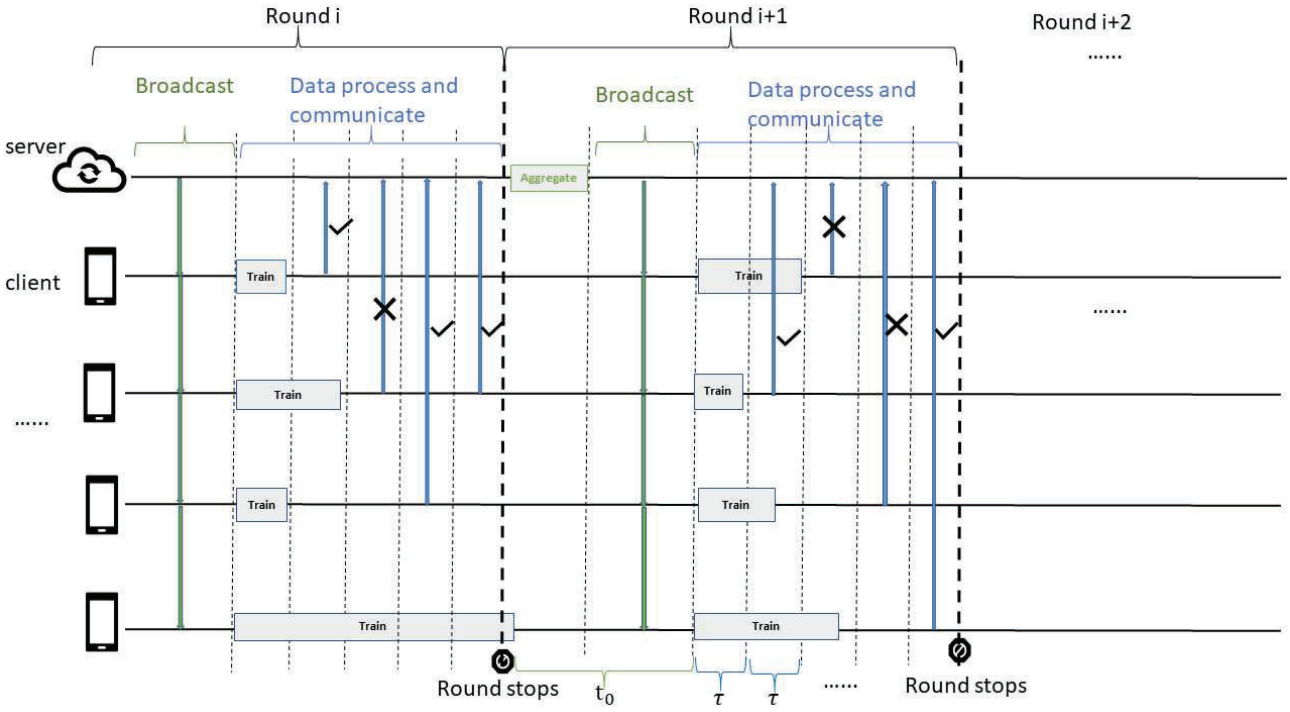

Fig. 1: Iterative training process

Data Processing Model: We assume each client finishes the data processing and computing the update with some probability p at each time slot. In other words, we assume the processing time of each dataset is geometrically distributed.

Communication Model: We assume the client mobile devices share a single channel when communicating with the parameter server (this assumption can be easily extended to multichannel OFDM systems). At the beginning of a time slot, one of the clients who have finished their computing tasks but have not transmitted the data to the parameter server will be selected uniformly at random to upload the update to the server, and the transmission succeeds with probability μ at the end of the time slot. At some stopping time (the choice of the stopping time is the focus of this paper), the parameter server stops accepting new updates and updates the global model using all uploaded information. The parameter server then broadcasts a new global model to all clients to start a new round.

This iterative training process is shown in Figure 1, where τ denotes length of a time slot. We assume the amount of time it takes for the parameter server to broadcast the updated parameters is t_0, which remains a constant for all rounds and includes both the time it takes to aggregate all the updates it receives and the time it takes to transmit the new parameter to all clients.

We define α_n to be the number of clients who have not finished processing their local datasets, β_n to be the number of clients who have finished computing but have not transmitted the updates to the parameter server, and $k_n = M - \alpha_n - \beta_n$ to be the number of clients who have updated the parameters based on local datasets and also uploaded the updates to the parameter server. The iterative training process can be modeled as a discrete-time Markov chain (DTMC) whose state at time slot n is denoted by $X_n = (K_n, \boldsymbol{\beta}_n)$. Let $x_n = (k_n, \beta_n)$ denote a realization of X_n.

Let $R(k)$ denote the reward that the parameter server obtains after receiving the kth update. The reward R can be decrement of the loss function. We make the following assumptions in this paper:

(1) R is positive, increasing with k and bounded.
(2) $\Delta R(k) = R(k+1) - R(k)$ is decreasing in k, i.e. the reward increment of receiving one more update is diminishing as the parameter receives more and more updates.

We first focus on a single round with a given reward function. Let N denote the number of time slots in this ground, which is a random variable as the parameter server can decide to terminate this round and start the next round at anytime. We consider the following stopping problem:

$$J^* = \sup_{\pi \in S} \frac{\mathbb{E}[R(K_N)]}{\mathbb{E}[N\tau + t_0]} \tag{1}$$

where π is a stopping policy, $\mathcal{S}$ is the set of all stopping policy, N is stopping time, and t_0 is a constant as defined above.

The problem above is difficult to solve. Instead, we introduce the following problem and then show that resolving this new problem can lead to the solution of the original problem:

$$V_\lambda = \sup_{\pi \in \mathcal{S}} \mathbb{E}\left[R(K_N) - \lambda(N\tau + t_0)\right] \tag{2}$$

where λ is a positive constant.

For simplicity, we include the time slot in the system state $(X_n, n) = (K_n, \boldsymbol{\beta}_n, n)$. The transition probabilities of the Markov chain are summarized below. Given state $(X_n, n) = (k, \beta, n)$ and any $0 \le i \le M - k - \beta$, we have

- If $\beta > 0$, then

$$\begin{aligned}&\Pr\left[(X_{n+1}, n+1) = (k, \beta + i, n+1) | (X_n, n) = (k, \beta, n)\right] \\ =&(1-\mu)\binom{M-k-\beta}{i} p^i (1-p)^{M-k-\beta-i},\end{aligned}$$

$$\begin{aligned}&\Pr\left[(X_{n+1}, n+1) = (k+1, \beta - 1 + i, n+1)\right. \\ &\qquad\qquad \left. | (X_n, n) = (k, \beta, n)\right] \\ =&\mu\binom{M-k-\beta}{i} p^i (1-p)^{M-k-\beta-i}.\end{aligned}$$

- If $\beta = 0$, then

$$\begin{aligned}&\Pr\left[(X_{n+1}, n+1) = (k, i, n+1) | (X_n, n) = (k, \beta, n)\right] \\ =&\binom{M-k-\beta}{i} p^i (1-p)^{M-k-\beta-i}.\end{aligned}$$

We define $V(\cdot)$ to be the value function so that $V(k, \beta, n)$ is the value of state $(x_n, n) = (k, \beta, n)$, and

$$\begin{aligned}&V(k, \beta, n) \\ =&\sup_{\pi, K_N \le M-k} \mathbb{E}\left[R(k + K_N) - \lambda((N+n)\tau + t_0)\right].\end{aligned}$$

We can easily verify that $V_\lambda = V(0, 0, 0)$.

The following theorem establishes the relationship between the original problem and the modified problem. (The proof of following theorems and lemmas can be found in our technical report [8].)

Theorem 1. *If there exists λ such that*

$$V_\lambda = \sup_{\pi \in S} \mathbb{E}(R(K_N) - \lambda(N\tau + t_0)) = 0,$$

then

$$J^* = \sup_{\pi \in S} \frac{\mathbb{E}[R(K_N)]}{\mathbb{E}[\lambda(N\tau + t_0)]} = \lambda.$$

Furthermore, if

$$V_\lambda = \sup_{\pi \in S} \mathbb{E}(R(K_N) - \lambda(N\tau + t_0)) = 0$$

is attained by some policy $\pi^* \in S$, then the policy π^* is also optimal for maximizing $\mathbb{E}\left[R(K_N)\right] / \mathbb{E}\left[N\tau + t_0\right]$.

Motivated by the result above, we can find the optimal stopping time for problem 1 by the following steps:

- First initialize λ, and find optimal π_λ in Problem (2) as well as V_λ.
- Repeatedly update λ, and find a new optimal policy π_λ and optimal value V_λ for Problem (2) until finding a λ^* such that optimal value $V_{\lambda^*} = 0$. The final policy π^* is the optimal policy.

In the next section, we present a low complexity algorithm for Problem (2).

III. Low-Complexity Algorithm for Solving the Modified Optimal Stopping problem

In this section, we focus on solving the optimal stopping time problem for a fixed λ. We will show that optimal policy is a threshold policy. For any state $(x_n, n) = (k, \beta, n)$, the actions are to either terminate the current round or to continue to the next time slot. The reward of stopping at that stage is $R(k) - \lambda(n\tau + t_0)$. If $\beta = 0$, then to continue means to let unfinished clients continue to compute, but no update will be transmitted since $\beta = 0$. Otherwise, $\beta_n \ge 1$, to continue means a randomly selected client from the β clients will transmit its result to the server while $(M-k-\beta)$ unfinished clients continue to process their datasets. The Bellman equation in these two cases are presented below.

If $\beta > 0$, then

$$\begin{aligned}&V(k, \beta, n) \\ =&\max\Big\{R(k) - \lambda(n\tau + t_0)\ , \\ &\qquad \mu\left(\sum_{i=0}^{M-k-\beta} W(i) V(k+1, \beta - 1 + i, n+1)\right) \\ &\qquad +(1-\mu)\left(\sum_{i=0}^{M-k-\beta} W(i) V(k, \beta + i, n+1)\right)\Big\}. \quad (3)\end{aligned}$$

If $\beta = 0$, then

$$\begin{aligned}&V(k, 0, n) \\ =&\max\left\{R(k) - \lambda(n\tau + t_0), \sum_{i=0}^{M-k-\beta} W(i) V(k, i, n+1)\right\}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4)\end{aligned}$$

where $W(i) = \binom{M-k-\beta}{i} p^i (1-p)^{M-k-\beta-i}$.

Theorem 2. *The following threshold policy is optimal for Problem* (2)*: For any state (k, β, n), policy π^* first check β,*

- *When $\beta > 0$: if $k < k^*$, continue; and if $k \ge k^*$, stop;*
- *When $\beta = 0$: if $k < k_0^*$, continue; and if $k \ge k_0^*$, stop.*

In the algorithm above, $k^* = \min\{k_1^*, M\}$, $k_1^* = \inf\{k : \Delta R(k) \le \frac{\lambda\tau}{\mu}\}$, and $k_0^* \le k^*$.

We remark that k^* has a closed-form expression but k_0^* does not. We next discuss how to calculate k_0^* numerically based on the following lemma.

Lemma 1. *Given any $\lambda > 0$, and any state (k, β, n) such that $k \le k^*$, $k + \beta \ge k^*$ and $n \ge k$, we have*

$$V(k, \beta, n) = R(k^*) - \lambda(n\tau + t_0) - (k - k^*)\frac{\lambda\tau}{\mu}.$$

From the proof of theorem 2, we can omit n in state(k, β, n) and just need to calculate a two-dimensional value table of (k, β) with a fixed n. We can calculate this value using dynamic programming and setting a fixed n larger than M. We maintain a value table of size of $M \times M$. The value of states

with different n can be calculated directly from following Lemma:

Lemma 2. *For any state (k,β,n) with $n \geq k$, $V(k,\beta,n+1) = V(k,\beta,n) - \lambda\tau$.*

Furthermore, the value for states with $k \geq k^*$and $n \geq k$ is shown from proof of theorem 2, and the value for states with $k \leq k^*$, $k+\beta \geq k^*$ and $n \geq k$ has been shown by lemma 1, so that we need to calculate a value table with size $k^* \times k^*$ instead of size $M \times M$, which are value of states with $k+\beta < k^*$. So we only need to use dynamic programming to calculate k_0^* as well as V_λ ($V(0,0,0)$).

We start from state $(k^*-1,0,n)$, where n is a fixed number larger than M. The Bellman equation is as follows:

$$\begin{aligned} &V(k^*-1,0,n) \\ &= \max\{R(k^*-1) - \lambda(n\tau+t_0), \\ &\qquad \sum_{i=0}^{M-k^*+1} G(i)\cdot V(k^*-1,i,n) - \lambda\tau \Bigg\}, \end{aligned} \tag{5}$$

where $G(i) = \binom{M-k^*+1}{i} p^i (1-p)^{M-k^*+1-i}$.

Since $V(k^*-1, i \geq 1, n)$ are known, and

$$V(k^*-1, i\geq 1, n) = R(k^*) - \lambda(n\tau+t_0) - \lambda\tau/\mu$$

according to Lemma 1, we can solve this Bellman equation and get $V(k^*-1,0,n)$.

Similarly we calculate the values of state $(k^*-2,1,n)$ and $(k^*-2,0,n)$. The optimal rule for state $(k^*-2,1,n)$ is to continue according to theorem 2, so

$$\begin{aligned} &V(k^*-2,1,n) \\ &= \mu\left(\sum_{i=0}^{M-k^*+1} G(i) V(k^*-1,i,n+1)\right) \\ &\quad + (1-\mu)\left(\sum_{i=0}^{M-k^*+1} G(i) V(k^*-2,1+i,n)\right) - \lambda\tau, \end{aligned} \tag{6}$$

where $G(i) = \binom{M-k^*+1}{i} p^i (1-p)^{M-k^*+1-i}$. All values on the right hand side except $V(k^*-2,1,n)$ are known, so $V(k^*-2,1,n)$ is done. For $(k^*-2,0,n)$, we have :

$$\begin{aligned} &V(k^*-2,0,n) \\ &= \max\{R(k^*-2) - \lambda(n\tau+t_0), \\ &\qquad \sum_{i=0}^{M-k^*+2} H(i)\cdot V(k^*-2,i,n) - \lambda\tau \Bigg\}, \end{aligned} \tag{7}$$

where $H(i) = \binom{M-k^*+2}{i} p^i (1-p)^{M-k^*+2-i}$. Since we just get $V(k^*-2,1,n)$, and $V(k^*-2, i\geq 2, n)$ are known by Lemma 1, after solving this bellman equation, we can get $V(k^*-2,\beta=0,n)$.

Next we can continue to calculate $V(k^*-3,2,n)$,$V(k^*-3,1,n)$, $V(k^*-3,0,n)$; and then $V(k^*-4,3,n)$,$V(k^*-4,2,n)$, $V(k^*-4,1,n)$,$V(k^*-4,0,n)$ $\cdots$, $V(0,\beta=k^*-1,n), V(0,\beta=k^*-2,n), ..., V(0,\beta=0,n)$. Now we have got this $k^* \times k^*$ value table.

According to Lemma 2, we can get $V_\lambda = V(0,0,0) = V(0,0,n) + n\lambda\tau$. From theorem 2 and principle of optimality, we can get

$$k_0^* = \min\left\{M, \inf\{k : V(k,0,n) = R(k) - \lambda(n\tau+t_0)\}\right\},$$

which determines the optimal stopping rule.

The algorithm is summarized below for given $\lambda > 0$.

Algorithm 1: The numerical algorithm for solving Problem 2

Given parameters: M,t_0,τ,λ, μ, p and reward function $R(k)$;
Calculate k^*, where $k^* = \min\{M, k_1^*\}$, $k_1^* = inf\{k : \Delta R(k) \leq \frac{\lambda\tau}{\mu}\}$
Set $n = k_1^* + 1$;
Get $V(k,\beta,n) = R(k) - \lambda(n\tau+t_0)$ directly by lemma 2, where $k \geq k^*, 0 \leq \beta \leq M-k^*$;
Get $V(k,\beta,n) = R(k^*) - \lambda(n\tau+t_0) - (k^*-k)\lambda\tau/\mu$ by lemma 5, where $k < k^*, k^*-k \leq \beta \leq M-k$;
Calculate $V(k^*-1,\beta=0,n)$ by solving the Bellman equation;
Calculate $V(k^*-2,\beta=1,n), V(k^*-2,\beta=0,n)$;
$V(k^*-3,\beta=2,n), V(k^*-3,\beta=1,n)$,
$V(k^*-3,\beta=0,n)$;
...
$V(0,\beta=k^*-1,n), V(0,\beta=k^*-2,n)$,
$..., V(0,\beta=0,n)$ sequentially;
Return optimal value $V_\lambda = V(0,0,0) = V(0,0,n) + \lambda n$;
Calculate $k_0^* = inf\{k : V(k,0,n) = R(k) - \lambda(n\tau+t_0)\}$,
Return optimal policy π_λ by theorem 2.
If $\beta > 0$: If $k < k^*$, continue; else, stop
else: If $k < k_0^*$: continue; else, stop.

Next we present a lower bound on J^* and an upper bound on k^*.

Corollary 1. *J^* is lower bounded by*

$$\frac{R(1)}{(\frac{1}{\mu}+\frac{1}{p})\tau + t_0}.$$

Proof. By the definition of J^*, we choose a policy π such that it stops when $k=1$. Therefore,

$$J^* \geq \frac{R(1)}{\mathbb{E}_\pi(N\tau+t_0)}.$$

If $M=1$, then $\mathbb{E}_\pi(N) = \frac{1}{\mu}+\frac{1}{p}$. Therefore, for any $M \geq 1$, $\mathbb{E}_\pi(N) \leq \frac{1}{\mu}+\frac{1}{p}$. Hence,

$$J^* \geq \frac{R(1)}{\mathbb{E}_\pi(N\tau+t_0)} \geq \frac{R(1)}{(\frac{1}{\mu}+\frac{1}{p})\tau+t_0}.$$

□

Corollary 2. *J^* is upper bounded by $\max_k \frac{R(k)}{k\tau+t_0}$, where k is an integer and $0 \le k \le M$.*

Proof. A straightforward upper bound on J^* is

$$J' = \sup_{\pi \in S} \frac{\mathbb{E}[R(K_N)]}{\mathbb{E}[N \cdot \tau + t_0]},$$

where p and μ are both set to 1, so J^* is upper bounded by $\max_k \frac{R(k)}{k\tau+t_0}$, where k is integer and $0 \le k \le M$. □

IV. The Solution of the Original Problem

In the previous section, we have shown how to find optimal π_λ for Problem (2) as well as V_λ for a given $\lambda > 0$. Next we focus on finding the optimal λ so that we can solve the original problem. Corollaries 1 and 2 show lower and upper bound on J^* , we can use them to initialize the λ. The following lemma demonstrates some important properties of V_λ.

Lemma 3. *V_λ is decreasing and convex in λ.*

Proof. The proof follows the analysis in [9]. Assuming $\lambda_1 < \lambda_2$, we have

$$\begin{aligned} V_{\lambda_2} &= E_{\pi_{\lambda_2}}[R(K_N) - \lambda_2(N\tau + t_0)] \\ &< E_{\pi_{\lambda_2}}[R(K_N) - \lambda_1(N\tau + t_0)] \\ &\le E_{\pi_{\lambda_1}}[R(K_N) - \lambda_1(N\tau + t_0)] \\ &= V_{\lambda_1}, \end{aligned}$$

so V_λ is decreasing with λ.

To prove convexity, given λ_1 and λ_2, let $0 < \theta < 1$, $\lambda = \theta\lambda_1 + (1-\theta)\lambda_2$, so

$$\begin{aligned} V_\lambda &= E_{\pi_\lambda}[R(K_N) - (\theta\lambda_1 + (1-\theta)\lambda_2)(N\tau + t_0)] \\ &= \theta E_{\pi_\lambda}[R(K_N) - \lambda_1(N\tau + t_0)] \\ &\quad + (1-\theta)E_{\pi_\lambda}[R(K_N) - \lambda_2(N\tau + t_0)] \\ &\le \theta V_{\lambda_1} + (1-\theta)V_{\lambda_2}. \end{aligned}$$

□

Now we can get an upper bound on k^* for all possible λ when $R(k) = c - \frac{a}{k+1}$ for some constants a and c.

Corollary 3. *Given $R(k) = c - \frac{a}{k+1}$, we have $k^* < \sqrt{\frac{a}{c-a/2}(1 + \frac{\mu}{p} + \frac{\mu t_0}{\tau})}$.*

Proof. It is directly from the definition of k^* and Corollary 1. Since $k^* = \inf\{k : \Delta R(k) \le \frac{\lambda\tau}{\mu}\}$, we have

$$\frac{a}{k^*(k^*+1)} > \lambda\tau/\mu > \frac{c - a/2}{(\frac{1}{\mu} + \frac{1}{p})\tau + t_0}\tau/\mu.$$

Therefore,

$$k^* < \sqrt{\frac{a}{c-a/2}(1 + \frac{\mu}{p} + \frac{\mu t_0}{\tau})}.$$

□

Corollary 3 establishes an upper bound on k^*, which depends on parameters a, c, μ, p, t_0, and τ, but independent of M. Therefore, the complexity of solving the modified problem (2) is $O(M(k^*)^2) = O(M)$.

The next algorithm presents the details of solving the optimal stopping time, where σ is the predefined accuracy level.

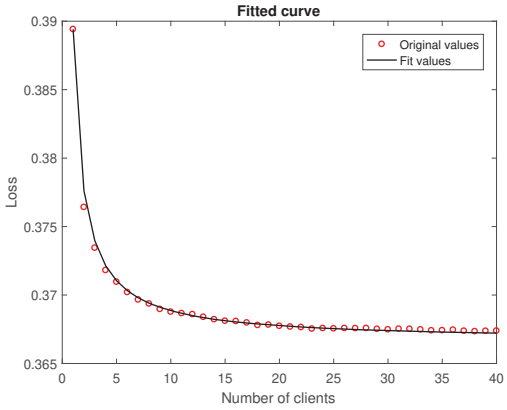

Fig. 2: simulation result about reward function(loss function) with 100 user

Algorithm 2: Solution for problem 1

Given parameters: M,t_0,τ, μ, p ; Reward function $R(k)$;
Step 1: Calculate lower bound λ_{lower} and upper bound λ_{upper} by corollary 1 and corollary 2.
Step 2: $\lambda = (\lambda_{upper} + \lambda_{lower})/2$
while $|V_\lambda| > \sigma = 0.001$ **do**
  if $V_\lambda > 0$ **then**
    $\lambda_{lower} \leftarrow \lambda$, $\lambda \leftarrow (\lambda_{upper} + \lambda_{lower})/2$
  else
    $\lambda_{upper} \leftarrow \lambda$, $\lambda \leftarrow (\lambda_{upper} + \lambda_{lower})/2$
  end
end
Step 4: return λ, which is equal to J^*

V. Evaluation

Data and Model: We consider the experiment of training a CNN model with distributed MNIST data. The dataset is divided into 100 groups, each representing a local dataset (or a client). Each client trains the CNN model with its own data and uploads its newly trained parameters sequentially to a parameter server if the channel is ON.

Reward function: We first plot the reward function $R(k)$ which is defined to be the decrement of the loss function (the cross-entropy loss) when the number of updates increases from $k-1$ to k. The loss function for $k = 1, \cdots, 40, ...$ is shown in Fig. 2, from which we can see that $R(k) = c - \frac{a}{k+1}$ fits the reward function well. So in our experiments, we assume $R(k) = c - \frac{a}{k+1}$ for some $a > 0$ and $c > 0$.

We evaluated the proposed algorithms using the MNIST dataset. In our experiment, we chose $M = 100$, $t_0 = 3,000$ms

which includes broadcasting time and aggregating time, $\tau = 10$ms, and defined the reward function to be $R(k) = 0.04 - \frac{0.018}{k+1}$.

We further chose success probability of transmissions μ to be $\frac{5}{8}$. The size of the parameters of our CNN is about 80k-100k. Based on the transmission rate of current 4G systems, which is about 50k per 10ms, we assume the average time for finishing uploading the parameters is around 16ms, which leads to our choice of μ. The success probability of data processing p is set to be $1/500$. We estimate the average training time on cell phone is $5000ms$. Since the duration of each time slot is $10ms$, the transmission probability is set to be $1/500$.

We compared the loss function under the proposed algorithm based on optimal stopping time and other algorithms based on fixed number of updates in each round. In particular, we considered two other algorithms: the first algorithm require updates from all M devices and the second algorithm, used by Google, requires 10% updates from the M device. We remark that the 10% rule is selected by comparing different fractions and found the best one for each application [10]. So it can be viewed as a policy uses the "optimal" number of updates at each iteration.

For the optimal stopping time algorithm, we first obtain lower bound $\lambda_{\text{lower}} = 0.01203$ and upper bound $\lambda_{\text{upper}} = 0.01234$. We then found the optimal $\lambda^* = 0.01209$, from which, we obtained that k^*=10 and $k_0^* = 8$ for the optimal stopping rule.

The testing loss as a function time is shown in Figure 3. Each data point in lines represents the test loss after one round of training. The length interval between two data points in each line of our figure shows average running time in one round for each stopping rule. For example, the average simulated running time of a round with the optimal stopping rule (k^* = 10, k_0^* = 8) is 3.45s, as well as $3.55s$ for $k = 10$ and 28.76s for $k = 100$. We can see that the optimal stopping rule reduce the loss by at least 170% throughout the training process. Figure 4 compares just the optimal stopping rule and the 10% rule. We again can see from this figure that even comparing to the "optimal" fixed k, the optimal stopping rule still reduces the loss function by 7% throughout.

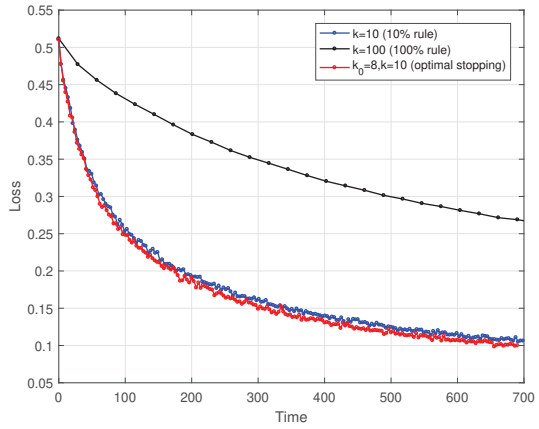

Fig. 3: Experiment result using optimal stopping rule with 100 users

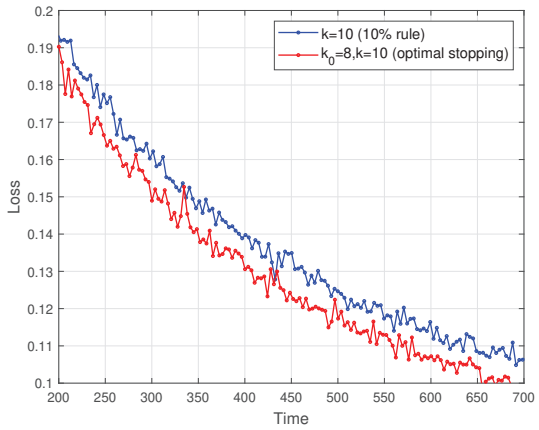

Fig. 4: Experiment result using optimal stopping rule with 100 users

VI. Conclusions

In this paper, we proposed an optimal stopping approach for training machine learning models in Federated Learning. Our simulation results showed the significant performance gain compared with fixed training schedules.

Acknowledgement

Research supported in part by NSF CNS 1618768, ECCS 1739344, and CNS 2002608.

References

[1] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):12, 2019.

[2] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.

[3] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[4] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[6] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *arXiv preprint arXiv:1810.08313*, 2018.

[7] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. *arXiv preprint arXiv:1808.07576*, 2018.

[8] P. Jiang and L. Ying. An optimal stopping approach for iterative training in federated learning. *Arizona State Technical Report*, 2019.

[9] Thomas S Ferguson. Optimal stopping and applications. 2012.

[10] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.