FDML: A Collaborative Machine Learning Framework for Distributed Features

Yaochen Hu[1], Di Niu[1], Jianming Yang[2], Shengping Zhou[2]
[1]University of Alberta, Edmonton, AB, Canada
[2]Platform and Content Group, Tencent, Shenzhen, China

ABSTRACT

Most current distributed machine learning systems try to scale up model training by using a data-parallel architecture that divides the computation for different samples among workers. We study distributed machine learning from a different motivation, where the information about the same samples, e.g., users and objects, are owned by several parities that wish to collaborate but do not want to share raw data with each other. We propose an asynchronous stochastic gradient descent (SGD) algorithm for such a feature distributed machine learning (FDML) problem, to jointly learn from distributed features, with theoretical convergence guarantees under bounded asynchrony. Our algorithm does not require sharing the original features or even local model parameters between parties, thus preserving the data locality. The system can also easily incorporate differential privacy mechanisms to preserve a higher level of privacy. We implement the FDML system in a parameter server architecture and compare our system with fully centralized learning (which violates data locality) and learning based on only local features, through extensive experiments performed on both a public data set *a*9a, and a large dataset of 5, 000, 000 records and 8700 decentralized features from three collaborating apps at Tencent including *T*encent MyApp, *T*ecent QQ Browser and *T*encent Mobile Safeguard. Experimental results have demonstrated that the proposed FDML system can be used to significantly enhance app recommendation in Tencent MyApp by leveraging user and item features from other apps, while preserving the locality and privacy of features in each individual app to a high degree.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Supervised learning**; • **Security and privacy** → *Privacy-preserving protocols.*

KEYWORDS

Distributed Machine Learning; Model Parallelism; Distributed Features; Stochastic Gradient Descent; Data Privacy; Optimization

ACM Reference Format:
Yaochen Hu, Di Niu, Jianming Yang, Shengping Zhou. 2019. FDML: A Collaborative Machine Learning Framework for Distributed Features. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330765

1 INTRODUCTION

While the success of modern machine learning lays the foundation of many intelligent services, the performance of a model often depends on the availability of data. In most applications, however, a large quantity of useful data may be generated on and held by multiple parties. Collecting such data to a central site for training incurs extra management and business compliance overhead, privacy concerns, or even regulation and judicial issues. As a result, a number of distributed machine learning techniques have been proposed to collaboratively train a model by letting each party perform local model updates and exchange locally computed gradients [27] or model parameters [21] with the central server to iteratively improve model accuracy. Most of the existing schemes, however, fall into the range of *data parallel* computation, where the training samples are located on different parties. For example, different users hold different images to jointly train a classifier. Different organizations may contribute their individual corpora to learn a joint language model.

We study distributed machine learning based on another motivation, where different *features* of a same sample are held by different parties. The question is—can we improve the predictive power at one party by leveraging additional features from another domain or party, yet without requiring any party to share its features? This is a real problem we are solving at *Tencent MyApp*, one of the largest Android app stores in China, with a market share of 24.7% in China in 2017. *Tencent MyApp* performs app recommendation and activation rate prediction based on the user and app features logged in its own platform. However, it turns out other Tencent apps including *Tencent QQ Browser* and *Tencent Mobile Safeguard* share a large number of common users with MyApp. Since these apps may have complementary information about a user, such cross-domain knowledge from another app, if utilized, may help to train a joint model that can improve app recommendation and customer behavior preference prediction in MyApp. However, due to privacy and customer protection regulations, raw customer data are not to be shared across apps that belong to different departments.

A natural question is—how can we train a joint machine learning model if the features of each training sample are located on multiple distributed parties? To make the solution practical with the most conservative assumption on information sharing, we bear the following goals:

- To minimize information leakage, no party should share its feature set. Neither should any of its local model parameters be communicated to other parties.

KDD ’19, August 4–8, 2019, Anchorage, AK, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330765

- The prediction made by the joint model should outperform the prediction made by each isolated model trained only with a single party's feature set, provided that such improvement from joint features also exists in centralized training.
- The joint model produced should approach the model trained in a centralized manner if all the features were collected centrally.
- The system should be efficient in the presence of both large numbers of features and samples.

To solve the above challenges, in this paper, we design, implement and extensively evaluate a practical Feature Distributed Machine Learning (FDML) system based on real-world datasets. For any supervised learning task, e.g., classification, our system enables each party to use an arbitrary model (e.g., logistic regression, factorization machine, SVM, and deep neural networks) to map its local feature set to a local prediction, while different local predictions are aggregated into a final prediction for classification via a "hyperlinear structure," which is similar to softmax. The entire model is trained end-to-end using a mini-batched stochastic gradient descent (SGD) algorithm performed in the sense of stale synchronous parallel (SSP) [13], i.e., different parties are allowed to be at different iterations of parameter updates up to a bounded delay.

A highlight of our system is that during each training iteration, every party is solely responsible for updating its own local model parameters (local net) using its own mini-batch of local feature sets, and for each record, only needs to share its local prediction to the central server (or to other parties directly in a fully decentralized scenario). Since neither the original features nor the local model parameters of a party are transferred to any external sites, the FDML system preserves data locality and is much less vulnerable to model inversion attacks [12] targeting other collaborative learning algorithms [27, 31] that share model parameters between parties. Moreover, we further enhance the data privacy by adopting a differential-privacy-based method [9, 10, 27]. by adding some perturbations to the shared local predictions.

We theoretically establish a convergence rate of $O(\frac{1}{\sqrt{T}})$ for the proposed asynchronous FDML algorithm under certain assumptions (including the bounded delay assumption [13]), where T is the number of iterations on (the slowest) party, which matches the standard convergence rate of fully centralized synchronous SGD training with a convex loss as well as that known for asynchronously distributed data-parallel SGD in SSP [13].

We developed a distributed implementation of FDML in a parameter server architecture, and conducted experiments based on both a public data set $a9$a [8], and a large dataset of $5,000,000$ samples and 8700 decentralized features collected from three popular Tencent Apps, including *Tencent MyApp*, *Tencent QQ Browser* and *Tencent Mobile Safeguard*. Extensive experimental results have demonstrated that FDML can even closely approach centralized learning in terms of testing errors, without violating data locality constraints, although centralized learning can use a more sophisticated model, since all features are collected centrally. In the meantime, FDML significantly outperforms models trained only based on the local features of each single app, demonstrating its advantage in harvesting insights from additional cross-domain features.

2 RELATED WORK

Distributed Machine Learning. Distributed machine learning algorithms and systems have been extensively studied in recent years to scale up machine learning in the presence of big data and big models. Existing work focuses either on the theoretical convergence speed of proposed algorithms, or on the practical system aspects to reduce the overall model training time [30]. Bulk synchronous parallel algorithms (BSP) [7, 32] are among the first distributed machine learning algorithms. Due to the hash constraints on the computation and communication procedures, these schemes share a convergence speed that is similar to traditional synchronous and centralized gradient-like algorithms. Stale synchronous parallel (SSP) algorithms [13] are a more practical alternative that abandons strict iteration barriers, and allows the workers to be off synchrony up to a certain bounded delay. The convergence results have been developed for both gradient descent and SGD [13, 20, 25] as well as proximal gradient methods [18] under different assumptions of the loss functions. In fact, SSP has become central to various types of current distributed Parameter Server architectures [1, 5, 14, 17, 19].

Depending on how the computation workload is partitioned [30], distributed machine learning systems can be categorized into *data parallel* and *model parallel* systems. Most of existing distributed machine learning systems [1, 5, 14, 17, 19] fall into the range of *data parallel*, where different workers hold different training samples.

Model Parallelism. There are only a couple of studies on *model parallel* systems, i.e., DistBelief [6] and STRADS [16], which aims to train a big model by letting each worker be responsible for updating a subset of model parameters. However, both DistBelief and STRADS, require collaborating workers to transmit their local model parameters to each other (or to a server), which violates our non-leakage requirement for models and inevitably incurs more transmission overhead. Furthermore, nearly all recent advances on model parallel neural networks (e.g., DistBelief [6] and AMPNet [2]) mainly partition the network horizontally according to neural network layers with motivation to scale up computation to big models. In contrast, we study a completely vertical partition strategy based strictly on features, which is motivated by the cooperation between multiple businesses/organizations that hold different aspects of information about the same samples. Another difference is that we do not require transmitting the model parameters; nor any raw feature data between parties.

On a theoretical perspective of model parallel algorithm analysis, [31] has proposed and analyzed the convergence of a model parallel yet non-stochastic *proximal gradient* algorithm that requires passing model parameters between workers under the SSP setting. Parallel coordinate descent algorithms have been analyzed recently in [4, 26]. Yet, these studies focus on randomized coordinate selection in a *synchronous* setting, which is different from our setting where multiple nodes can update disjoint model blocks asynchronously. Although Stochastic gradient descent (SGD) is the most popular optimization method extensively used for modern distributed data analytics and machine learning, to the best of our knowledge, there is still no convergence result of (asynchronous) SGD in a model parallel setting to date. Our convergence rate of FDML offers the first analysis of asynchronous *model parallel* SGD,

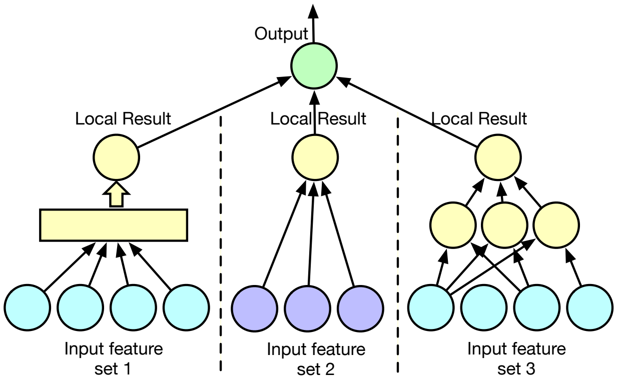


Figure 1: Illustrating FDML model (2)**, where the local predictions, each depending on the local model on a party, are aggregated into a final output using linear and nonlinear transformations** (1)**.**

which matches the standard convergence rate of the original SSP algorithm [13] for data parallel SGD.

Learning Privately. A variant of distributed SGD with a filter to suppress insignificant updates has recently been applied to collaborative deep learning among multiple parties in a data parallel fashion [27]. Although raw data are not transferred by the distributed SGD in [27], a recent study [12] points out that an algorithm that passes model parameters may be vulnerable to model inversion attacks based on generative adversarial networks (GANs). In contrast, we do not let parties transfer local model parameters to server or any other party.

Aside from the distributed optimization approach mentioned above, another approach to privacy preserving machine learning is through encryption, e.g., via homomorphic encryption [11, 28] or secret sharing [3, 22, 29]. Models are then trained on encrypted data. However, this approach cannot be flexibly generalized to all algorithms and operations, and incurs additional computation and design cost. Relatively earlier, differential privacy has also been applied to collaborative machine learning [23, 24], with an inherent tradeoff between privacy and utility of the trained model. To the best of our knowledge, none of the previous work addressed the problem of collaborative learning when the features of each training sample are distributed on multiple participants.

3 PROBLEM FORMULATION

Consider a system of m different parties, each party holding different aspects about the same training samples. Let $\{(\xi_i^1, \xi_i^2, \ldots, \xi_i^m), y_i\}_{i=1}^n$ represent the set of n training samples, where the vector $\xi_i^j \in \mathbb{R}^{d^j}$ denotes the features of the ith sample located on jth party, and y_i is the label of sample i. Let $\xi_i \in \mathbb{R}^d$ be the overall feature vector of sample i, which is a concatenation of the vectors $\xi_i^1, \xi_i^2, \ldots, \xi_i^m$, with $d = \sum_j d_j$. Suppose the parties are not allowed to transfer their respective feature vector to each other out of regulatory and privacy reasons as has been mentioned above. In our problem, the feature vectors on two parties may or may not contain overlapped features. The goal of machine learning is to find a model $p(x, \xi)$ with parameters x that given an input ξ, can predict its label y, by minimizing the loss between the model prediction $p(x, \xi_i)$ and its corresponding label y_i over all training samples i.

We propose a Feature Distributed Machine Learning (FDML) algorithm that can train a joint model by utilizing all the distributed features while keeping the raw features at each party unrevealed to other parties. To achieve this goal, we adopt a specific class of model that has the form

$$p(x, \xi) = \sigma\Big(\sum_{j=1}^{m} a_j \alpha^j(x^j, \xi^j)\Big), \tag{1}$$

where $\alpha^j : \mathbb{R}^{D^j} \times \mathbb{R}^{d^j} \to \mathbb{R}$, $j = 1, \ldots, m$, is a sub-model on party j with parameters $x^j \in \mathbb{R}^{D^j}$, which can be a general function that maps the local features ξ^j on each party j to a local prediction. In addition, $\sigma : \mathbb{R} \to \mathbb{R}$ is a continuously differentiable function to aggregate local intermediate predictions $\alpha^j(x^j, \xi^j)$ weighted by a_j. Note that $x \in \mathbb{R}^D$, with $D = \sum_j D_j$, is a concatenation of the local model parameters x^j over all parties j.

As illustrated by Fig. 1, the model adopted here is essentially a composite model, where each sub-model α^j on party j with parameters x^j could be an arbitrary model, e.g., logistic regression, SVM, deep neural networks, factorization machines, etc. Each sub-model x^j on party j is only concerned with the local features ξ^j. The final prediction is made by merging the local intermediate results through a linear followed by nonlinear transformations, e.g., a softmax function. Note that in (1), all a_j can be eliminated by scaling some corresponding parameters in $\alpha(x^j, \xi^j)$ by $1/a_j$. Without loss of generality, we simplify the model to the following:

$$p(x, \xi) = \sigma\Big(\sum_{j=1}^{m} \alpha^j(x^j, \xi^j)\Big). \tag{2}$$

Apparently, in this model, both the local features ξ^j and the sub-model parameters x^j are stored and processed locally within party j, while only the local predictions $\alpha^j(x^j, \xi^j)$ need be shared to produce the final prediction. Therefore, the raw features as well as all sub-model parameters are kept private. In Sec. 4, we propose an asynchronous SGD algorithm that also preserves the non-sharing properties for all the local features as well as all sub-model parameters even during the model training phase, with theoretical convergence guarantees.

In general, the model is trained by solving the following problem:

$$\text{minimize}_x \frac{1}{n}\sum_{i=1}^{n} L(x; \xi_i, y_i) + \lambda \sum_{j=1}^{m} z^j(x^j), \tag{3}$$

where $L\big(p(x, \xi); y\big)$ is the loss function, indicating the gap between the predicted value and the true label for each sample. $z(x^j)$ is the regularizer for sub-model x^j.

4 ASYNCHRONOUS SGD FOR FDML

In this section, we describe our asynchronous and distributed stochastic gradient descent (SGD) algorithm specifically designed to solve the optimization problem (3) in FDML, with theoretical convergence guarantees.

Since we consider a stochastic algorithm, let $i(t)$ be the index of the sample $\xi_{i(t)}$ presented to the training algorithm in iteration t.

We denote the regularized loss of sample $i(t)$ by

$$F_t(x) := L(x; \xi_{i(t)}, y_{i(t)}) + \lambda \sum_{j=1}^{m} z^j(x^j). \tag{4}$$

In stochastic optimization, minimizing the loss in (3) over the entire training set is equivalently to solving the following problem [13]:

$$\text{minimize}_x F(x) := \frac{1}{T} \sum_t F_t(x), \tag{5}$$

where T is the total number of iterations. Let $\nabla F(x) \in \mathbb{R}^D$ be the gradient of F. Let $\nabla^j F(x) \in \mathbb{R}^{D^j}$ be the partial gradient of F with respect to the sub-model parameters $x^j \in \mathbb{R}^{D^j}$, i.e., $\nabla^j F(x) := \frac{\partial F(x)}{\partial x^j}$. Clearly, $\nabla F(x)$ is the concatenation of all the partial gradients $\nabla^1 F(x), \nabla^2 F(x), \ldots, \nabla^m F(x)$.

4.1 The Synchronous Algorithm

In a synchronous setting, we can simply parallelizing a SGD algorithm by updating each parameter block x^j concurrently for all $j = 1, \ldots, m$, given a coming sample $i(t)$, i.e., $x^j_{t+1} := x^j_t - \eta_t \nabla^j F_t(x^1_t, \ldots, x^m_t)$, where η_t is a predefined learning rate scheme. Specifically for model (2), according to (4), we can obtain the partial gradient $\nabla^j F_t(x)$ for $j = 1, \ldots, m$ as

$$\begin{aligned}
\nabla^j F_t(x) = {} & \lambda \frac{\partial z^j(x^j)}{\partial x^j} + \\
& L'\Big(\sigma\Big(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Big)\Big)\sigma'\Big(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Big)\frac{\partial \alpha^j(x^j, \xi^j_{i(t)})}{\partial x^j} \quad (6) \\
:= {} & H\Big(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Big)\frac{\partial \alpha^j(x^j, \xi^j_{i(t)})}{\partial x^j} + \lambda \frac{\partial z^j(x^j)}{\partial x^j}, \quad (7)
\end{aligned}$$

where we simplify the notation of the first few terms related to $\sum_{k=1}^m \alpha^k(x^k, \xi^k_{i(t)})$ by a function $H(\cdot)$. In practice, z^j could be nonsmooth. This setting is usually handled by proximal methods. In this work, we are only focused on the smooth case.

This indicates that for the class of models in (2) adopted by FDML, each party j does not even need other parties' models x^k, where $k \neq j$, to compute its partial gradient $\nabla^j F_t$. Instead, to compute $\nabla^j F_t$ in (7), each party j only needs one term, $\sum_{k=1}^m \alpha^k(x^k, \xi^k_{i(t)})$, which is the aggregation of the local prediction results from all parties at iteration t, while the remaining terms in (7) is only concerned with party j's local model x^j and local features $\xi^j_{i(t)}$. Therefore, this specific property enables a parallel algorithm with minimum sharing among parties, where neither local features nor local model parameters need be passed among parties.

4.2 The Asynchronous Algorithm

The *asynchronous* implementation of this idea in a distributed setting of multiple parties, with theoretical convergence guarantees, is significantly more challenging than it seems. As our proposed algorithm is closely related to asynchronous SGD, yet extends it from the data-parallel setting [13] to a block-wise model parallel setting, we would call our algorithm Asynchronous SGD for FDML.

Note that in an asynchronous setting, each party j will update its own parameters x^j_t asynchronously and two parties may be in different iterations. However, we assume different parties go through the samples $\xi_{i(t)}$ in the same order, although asynchronously, i.e., all the parties share the randomly generated sample index sequence $\{i(t) | t = 1, \ldots, T\}$, which can easily be realized by sharing the seed of a pseudo random number generator.

When each party j has its own iteration t, the local model parameters x^j_t on party j is updated by

$$x^j_{t+1} = x^j_t - \eta_t \Big(H\Big(\sum_{k=1}^{m} \alpha^k(x^k_{t-\tau^j_t(k)}, \xi^k_{i(t)})\Big)\frac{\partial \alpha^j(x^j_t, \xi^j_{i(t)})}{\partial x^j} + \lambda \frac{\partial z^j(x^j_t)}{\partial x^j}\Big), \tag{8}$$

where the requested aggregation of local predictions for sample $\xi_{i(t)}$ may be computed from possibly *stale versions* of model parameters, $x^k_{t-\tau^j_t(k)}$ on other parties $k \neq j$, where $\tau^j_t(k)$ represents how many iterations of a "lag" there are from party k to party j at the tth iteration of party j. We abuse the word "lag" here since party k could be faster than party j. We overflow the notation for that case by assigning negative value to $\tau^j_t(k)$. We give a convergence speed guarantee of the proposed algorithm under certain assumptions, when the lag $\tau^j_t(k)$ is bounded.

5 DISTRIBUTED IMPLEMENTATION

5.1 Implementation

We present a distributed implementation of the proposed asynchronous SGD algorithm for FDML. Our implementation is inspired by the Parameter Server architecture [5, 17, 18]. In a typical Parameter Server system, the workers compute gradients while the server updates the model parameters with the gradients computed by workers. Yet, in our implementation, as described in Algorithm 1, the only job of the server is to maintain and update a matrix $A_{i,j}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, which is introduced to hold the latest m local predictions for each sample i. We call $[A_{i,j}]_{n \times m}$ the *local prediction matrix*. On the other hand, unlike servers, the workers in our system each represent a participating party. They do not only compute gradients, but also need to update their respective local model parameters with SGD.

Furthermore, since each worker performs local updates individually, each worker can even further employ a parameter server cluster or a shared-memory system, e.g., a CPU/GPU workstation, to scale up and parallelize the computation workload related to any local model it adopts, e.g., a DNN or FM. A similar hierarchical cluster is considered in Gaia [14], though for data-parallel machine learning among multiple data centers.

First, we describe how the input data should be prepared for the FDML system. Before the training task, for consistency and efficiency, a *sample coordinator* will first randomly shuffle the sample indices and generate the sample presentation schedule $i(t)$, which dictates the order in which samples should be presented to the training algorithm. However, since features of a same sample are located on multiple parties, we need to find all the local features $\xi^1_i, \xi^2_i, \ldots, \xi^m_i$ as well as the label y_i associated with sample i. This can be done by using some common identifiers that are present in all local features of a sample, like user IDs, phone numbers, data

Algorithm 1 A Distributed Implementation of FDML

```
**Require**: each worker $j$ holds the local feature set $\{\xi_i^j, y_i\}_{i=1}^n$,
$j = 1, \ldots, m$; a sample presentation schedule $i(t)$, $t = 1, \ldots, T$, is
pre-generated randomly and shared among workers.
**Output**: model parameters $x_T = (x_T^1, \ldots, x_T^m)$.
Server:
    Initialize the local prediction matrix $[A_{i,j}]_{n \times m}$.
    while True do
        if Pull request (worker: $j$, iteration: $t$) received then
            if $t$ is not $\tau$ iterations ahead of the slowest worker then
                Send $\sum_{k=1}^m A_{i(t),k}$ to Worker $j$
            else
                Reject the Pull request
        if Push request (worker: $j$, iteration: $t$, value: $c$) received then
            $A_{i(t),j} := c$.
Worker $j$ ($j = 1, \ldots, m$) asynchronously performs:
    for $t = 1, \ldots, T$ do
        Push $c := \alpha^j(x_t^j, \xi_{i(t)}^j)$ to Server
        while Pull not successful do
            Pull $\sum_{k=1}^m A_{i(t),k}$ from Server
        $\nabla^j F_t := \left( H(\sum_{k=1}^m A_{i(t),k}) \cdot \frac{\partial \alpha^j(x_t^j, \xi_{i(t)}^j)}{\partial x^j} + \lambda \frac{\partial z^j(x_t^j)}{\partial x^j} \right)$
        Update the local weights as
            $x_{t+1}^j := x_t^j - \eta_t \nabla^j F_t.$    (9)
```

of birth plus name, item IDs, etc. Finally, the labels y_i will be sent to all workers (parties) so that they can compute error gradients locally. Therefore, before the algorithm starts, each worker j holds a local dataset $\{\xi_i^j, y_i\}_{i=1}^n$, for all $j = 1, \ldots, m$.

Let us explain Algorithm 1 from a worker's perspective.

To solve for x collaboratively, each worker j goes through the iterations $t = 1, \ldots, T$ individually and asynchronously in parallel, according to the (same) predefined sample presentation schedule $i(t)$ and updates its local model x^j according to (9). In a particular iteration t, when worker j updates x_t^j with the current local features $\xi_{i(t)}^j$, it first sends its updated local prediction about sample $i(t)$ to the server in order to update $A_{i(t),j}$, i.e., $A_{i(t),j} := \alpha^j(x_t^j, \xi_{i(t)}^j)$. And this update is done through the value c uploaded to the server in a *Push request* from worker j with iteration index t and value c. After this update, it pulls the latest $\sum_{k=1}^m A_{i(t),k}$ from the server based on the latest versions of local predictions, $A_{i(t),k}$, maintained on the server for all the workers $k = 1, \ldots, m$. Then x_t^j is updated into x_{t+1}^j locally by (9).

Since the workers perform local model updates asynchronously, at a certain point, different workers might be in different iterations, and a faster worker may be using the stale local predictions from other workers. We adopt a *stale synchronous* protocol to strike a balance between the evaluation time for each iteration and the total number of iterations to converge—a fully synchronous algorithm takes the least number of iterations to converge yet incurs large waiting time per iteration due to straggler workers, while on the other hand, an asynchronous algorithm reduced the per iteration evaluation time, at the possible cost of more iterations to converge. In order to reduce the overall training time, we require that the iteration of the fastest party should not exceed the iteration of the slowest party by τ, i.e., the server will reject a pull request if the t from the *Pull request*(worker: j, iteration: t) is τ iterations ahead of the slowest worker in the system. A similar bounded delay condition is enforced in most Parameter-Server-like systems [1, 5, 14, 17, 19] to ensure convergence and avoid chaotic behavior of a completely asynchronous system.

In real applications, the SGD algorithm can easily be replaced with the mini-batched SGD, by replacing the sample presentation schedule $i(t)$ with a set $I(t)$ representing the indices of a mini-batch of samples to be used iteration t, and replacing the partial gradient in (8) with the sum of partial gradients over the mini-batch $I(t)$.

5.2 Privacy

In FDML, one of the primary concerns is to preserve the privacy of the local feature data. Due to the specific model structure and the well designed algorithm, no model weights or features are uploaded from any parties. The only shared information is the intermediate local prediction results for each training or testing sample, which is some comprehensive function over both the local features and model weights. Therefore, there is little chance to leak the original features to honest servers or other parties.

To further protect the feature data at each party from malicious servers and parties, we apply differential privacy based methods by perturbing the local predictions to be uploaded [9, 10, 27]. In particular, we add some noise to the local prediction result $\alpha^j(x_t^j, \xi_{i(t)}^j)$ at party j to protect the privacy of all the input features at party j.

6 CONVERGENCE ANALYSIS

Inspired by a series of studies [13–15] on the convergence behavior of convex objective functions, we analyze the convergence property of the proposed asynchronous algorithm by evaluating a *regret* function R, which is the difference between the aggregated training loss and the loss of the optimal solution, defined as

$$R = \frac{1}{T} \sum_t F_t(x_t) - F(x_*), \tag{10}$$

where x_* is the optimal solution for $F(x)$, such that $x_* = \text{argmin}_x F(x)$. During training, the same set of data will be looped through for several *epochs*. This is as if a very large dataset is gone through till Tth iteration. We will prove convergence by showing that R will decrease to 0 with regard to T. Before presenting the main result, we introduce several notations and assumptions. We use D_t to denote the distance measure from x_t to x_*, i.e., $D_t := \frac{1}{2} \|x_t - x_*\|_2^2$. We make the following common assumptions on the loss function, which are used in many related studies as well.

Assumption 1. *(1) The function F_t is differentiable and the partial gradient $\nabla^j f$ are Lipschitz continuous with L_j, namely,*

$$\|\nabla^j F_t(x_1) - \nabla^j F_t(x_2)\| \leq L_j \|x_1 - x_2\|, \tag{11}$$

for $\forall x_1, x_2 \in \mathbb{R}^D$. We denote L_{max} as the maximum among the L_j for $\forall j$.

(2) Convexity of the loss function $F_t(x)$.

Table 1: The performance on Tencent MyApp data.

Algorithm	Train loss	Test loss	Test AUC	Time(s)
LR local	0.1183	0.1220	0.6573	546
LR centralized	0.1159	**0.1187**	**0.7037**	1063
LR FDML	**0.1143**	0.1191	0.6971	3530
NN local	0.1130	0.1193	0.6830	784
NN centralized	**0.1083**	0.1170	**0.7284**	8051
NN FDML	0.1101	**0.1167**	0.7203	4369

Table 2: The performance on a9a data.

Algorithm	Train loss	Test loss	Test AUC	Time(s)
LR local	0.3625	0.3509	0.8850	41
LR centralized	0.3359	0.3247	0.9025	45
LR FDML	**0.3352**	**0.3246**	**0.9026**	99
NN local	**0.3652**	0.3484	0.8864	53
NN centralized	0.4008	**0.3235**	**0.9042**	57
NN FDML	0.4170	0.3272	0.9035	110

(3) Bounded solution space. There exists a $D > 0$, s.t., $D_t \leq \frac{1}{2}D^2$ for $\forall t$.

As a consequence of the assumptions, the gradients are bounded, i.e., $\exists G > 0$, s.t., $\|\nabla F(x)\|_2^2 \leq G^2$ãĂĊ for $\forall x \in \mathbb{R}^D$ With these assumptions, we come to our main result on the convergence rate of the proposed SGD algorithm.

Proposition 1. *Under circumstances of the assumptions in Assumption 1, with a learning rate of $\eta_t = \frac{\eta}{\sqrt{t}}$, and a bounded staleness of τ, the regret R given by the updates* (8) *for the FDML problem is $R = O(\frac{1}{\sqrt{T}})$.*

Proof. Please refer to Appendix for the proof.

7 EXPERIMENTS

We are testing the application of the proposed FDML system in an app recommendation task at Tencent MyApp, which is a major Android market with an extremely large body of users in China. In this task, *user features*, including the past download activities in MyApp, are recorded. In the meantime, the task can also benefit from cross-domain features about the same users logged in two other apps (run by different departments of the same company), including QQ Browser that tracks user interests based on their content viewing history, as well as Tencent Mobile Safeguard, which records the app invoking and usage history of users.

The goal here is to leverage the additional user features available from the other domains to improve the app recommendation in MyApp, yet without having to download the raw user features from other apps to avoid regulatory issues, as customer data in different departments are protected under different security levels and even under different customer agreements. Some sensitive features under strong protection are prohibited to be moved to other parties, including other departments.

The dataset we use contains 5, 000, 000 labeled samples indicating whether a user will download an app or not. Each sample is a user-app pair, which contains around 8, 700 (sparse) features in total, among which around 7, 000 features come from Tencent MyApp itself, while the remaining 1, 700 features are from the other two apps. We randomly shuffle the data and split it into a 4.5 million training set and a 0.5 million testing set.

We also evaluate FDML on another public data set a9a [8], a classical census dataset, where the prediction task is to determine whether a person makes over $50K a year. There are 48, 842 samples, each with 124 features. 32, 661 samples are training data and 16, 281 samples are testing data. We split the 124 features into two sets of 67 and 57. We run both a logistic regression (LR) and a two layered fully connected neural network (NN) under three different training schemes for both data sets:

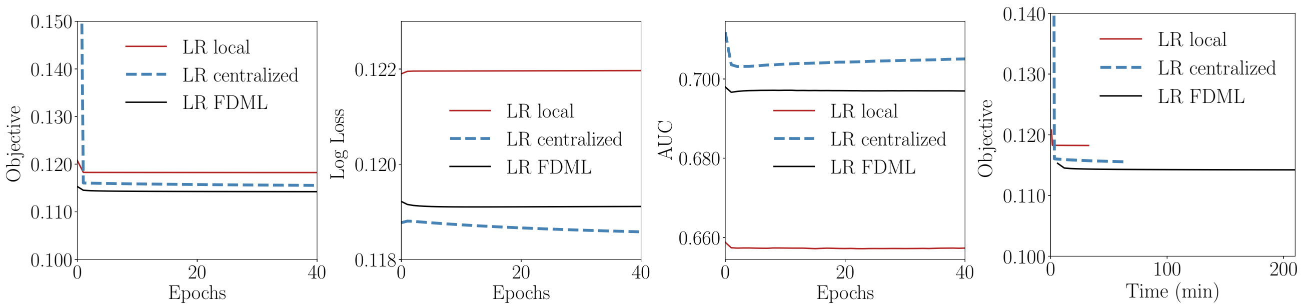
- *Local*: only use the 7, 000 local features from MyApp or the 67 features of a9a to train a model.
- *Centralized*: collect all the 8, 700 features from all three apps to a central server or using all the 124 features in a9a and train the model using the standard mini-batched SGD.
- *FDML*: use FDML system to train a joint model for app recommendation based on all 8, 700 features distributed in three apps or train the a9a classification model on all 124 features from two different parties, without centrally collecting data.

For FDML, there is a single server with several workers, each of which equipped with an Intel Xeon CPU E5-2670 v3 @ 2.30GHz. Each worker handles the features from one party. The system will be asynchronous as the lengths of features handled by each worker are different. The FDML NN only considers a fully connected NN within each party while merging the three local predictions in a composite model, whereas the Centralized NN uses a fully connected neural network over all the 8, 700124 features, thus leading to a more complex model (with interactions between the local features of different departments) than FDML NN.

For all training schemes, a mini-batched SGD is used with a batch size of 100. For each epoch, we keep track of the optimization objective value for training data, the log loss, the AUC for testing data and the elapsed time. Fig. 2 and Fig. 3 present the major statistics of the models during the training procedure for LR and NN for Tencent MyApp dataset, respectively. Table 1 presents the detailed statistics at the epoch when all the algorithms yield a stable and good performance on the testing data. Table 2 presents the performance for a9a dataset. The results show that FDML outperforms the corresponding Local scheme with only local features, and even approaches the performance of the Centralized scheme, while keeping the feature sets local to their respective workers.

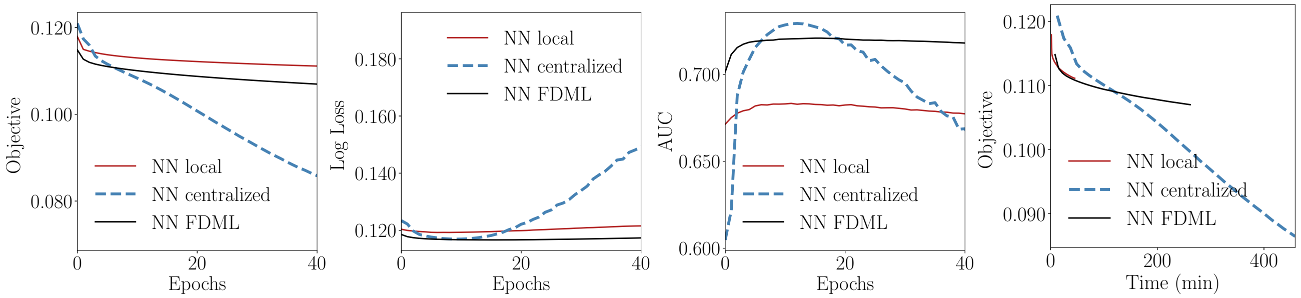
For LR, as shown by Fig. 2, Table 1 and Table 2, we can see that Centralized LR and FDML LR both achieve a smaller training objective value as well as significantly better performance on the testing set than Local LR. As we have expected, additional features recorded by other related services could indeed help improve the app recommendation performance. Furthermore, Centralizd LR and FDML LR have very close performance, since these two methods use the essentially the same model for LR, though with different training algorithms.

For NN shown in Fig. 3, Table 1 and Table 2, by leveraging additional features, both FDML NN and Centralized NN substantially outperform Local NN. Meanwhile, Centralized NN is slightly better than FDML NN, since Centralized NN has essentially adopted a more complex model, enabling feature interaction between different parties directly through fully connected neural networks.



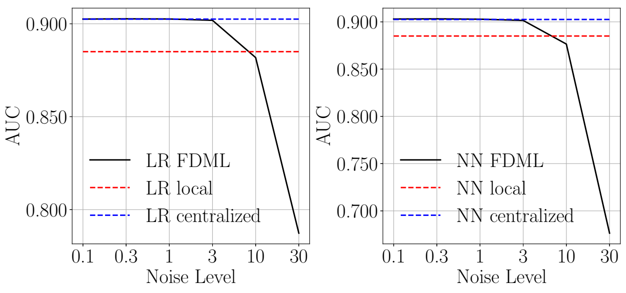
(a) Training objective vs. epoch (b) Tesiting log loss vs. epoch (c) Tesiting AUC vs. epoch (d) Training objective vs. time

Figure 2: A comparison between the three model training schemes for the LR model. All curves are plotted for epochs 1–40, including the time curve in (d).



(a) Training objective vs. epoch (b) Tesiting log loss vs. epoch (c) Tesiting AUC vs. epoch (d) Training objective vs. time

Figure 3: A comparison between the three model training schemes for the NN model. All curves are plotted for epochs 1–40, including the time curve in (d).



(a) Test AUC vs. added noise for LR (b) Test AUC vs. added noise for NN

Figure 4: Testing AUC under different levels of added noise during training for a9a data set

Fig. 2(d) and Fig. 3(d) compare the training time and speed among the three learning schemes for Tencent MyApp dataset. Without surprise, for both the LR and NN model, the Local scheme is the fastest since it uses the smallest amount of features and has no communication or synchronization overhead. For LR in Fig. 2(d), FDML LR is slower than Centralized LR since the computation load is relatively smaller in this LR model and thus the communication overhead dominates. On the contrary, for NN, as shown in Fig. 3(d), the Centralized NN is slower than FDML NN. This is because Centralized NN has much more inner connections and hence much more model parameters to train. Another reason is that FDML distributes the heavy computation load in this NN scenario to three different workers, which in fact speeds up training. Interestingly, for the smaller dataset a9a, in Table 2, the NN FDML is slower than the centralized one since in this case, the model is small and the communication overhead dominate the processing time in FDML.

Fig. 4 shows the performance when different levels of noise is added according to the differential privacy mechanism during the training procedure for a9a dataset. Conforming to the intuition, we can see that a higher level of noise will bring worse results. However, for a noise level no more than 3, we can still expect a performance improvement over learning only based on local data, while achieving stronger privacy guarantee due to the perturbations introduced to the shared local prediction results.

8 CONCLUSIONS

We study a feature distributed machine learning (FDML) problem motivated by real-world recommender applications at Tencent MyApp, where the features about the same training sample can be found at three different apps. However, the features of one app should be kept confidential to other parties due to regulatory constraints. This motivation is in contrast to most existing literature on collaborative and distributed machine learning which assumes the data samples (but not the features) are distributed and works in a data-parallel fashion. We propose an asynchronous SGD algorithm to solve the new FDML scenario, with a convergence rate of $O(1/\sqrt{T})$, T being the total number of iterations, matching the existing convergence rate known for data-parallel SGD in a stale synchronous parallel setting [13].

We have developed a distributed implementation of the FDML system in a parameter server architecture and performed extensive evaluation based on both a public data set and a large dataset of

5, 000, 000 records and 8, 700 decentralized features from *Tencent MyApp*, *Tencent QQ Browser* and *Tencent Mobile Safeguard* for a realistic app recommendation task. Results have shown that FDML can closely approximate centralized training (the latter collecting all data centrally and using a more complex model allowing more interactions among cross-domain features) in terms of the testing AUC, while significantly outperforming the models trained only based on the local features of *MyApp*. Currently, we are deploying the FDML system at *Tencent MyApp* and improving the robustness of the system by adding momentum based techniques. We are also developing schemes that can support more sophisicated models, taking more interactions between cross-party features into account.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large-Scale Machine Learning. In *Proc. USENIX Symposium on Operating System Design and Implementation (OSDI)*.

[2] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *arXiv preprint arXiv:1802.09941* (2018).

[3] Charlotte Bonte and Frederik Vercauteren. 2018. *Privacy-Preserving Logistic Regression Training.* Technical Report. IACR Cryptology ePrint Archive 233.

[4] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. 2011. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379* (2011).

[5] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

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

[7] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research* 13, Jan (2012), 165–202.

[8] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[9] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation.* Springer, 1–19.

[10] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[11] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning.* 201–210.

[12] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 603–618.

[13] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems.* 1223–1231.

[14] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds.. In *NSDI.* 629–647.

[15] John Langford, Alexander J Smola, and Martin Zinkevich. 2009. Slow learners are fast. *Advances in Neural Information Processing Systems* 22 (2009), 2331–2339.

[16] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. 2014. On model parallelization and scheduling strategies for distributed machine learning. In *Advances in neural information processing systems.* 2834–2842.

[17] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 14. 583–598.

[18] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems.* 19–27.

[19] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. 2016. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining.* ACM, 377–386.

[20] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems.* 2737–2745.

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

[22] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *2017 38th IEEE Symposium on Security and Privacy (SP).* IEEE, 19–38.

[23] Manas Pathak, Shantanu Rane, and Bhiksha Raj. 2010. Multiparty differential privacy via aggregation of locally trained classifiers. In *Advances in Neural Information Processing Systems.* 1876–1884.

[24] Arun Rajkumar and Shivani Agarwal. 2012. A differentially private stochastic gradient descent algorithm for multiparty classification. In *Artificial Intelligence and Statistics.* 933–941.

[25] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems.* 693–701.

[26] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. 2012. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems.* 28–36.

[27] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security.* ACM, 1310–1321.

[28] Hassan Takabi, Ehsan Hesamifard, and Mehdi Ghasemi. 2016. Privacy preserving multi-party machine learning with homomorphic encryption. In *29th Annual Conference on Neural Information Processing Systems (NIPS).*

[29] Li Wan, Wee Keong Ng, Shuguo Han, and Vincent Lee. 2007. Privacy-preservation for gradient descent methods. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 775–783.

[30] Eric P Xing, Qirong Ho, Pengtao Xie, and Dai Wei. 2016. Strategies and principles of distributed machine learning on big data. *Engineering* 2, 2 (2016), 179–195.

[31] Yi Zhou, Yaoliang Yu, Wei Dai, Yingbin Liang, and Eric Xing. 2016. On convergence of model parallel proximal gradient algorithm for stale synchronous parallel system. In *Artificial Intelligence and Statistics.* 713–722.

[32] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. 2010. Parallelized stochastic gradient descent. In *Advances in neural information processing systems.* 2595–2603.

9 APPENDIX

Proof of Proposition 1. By the proposed algorithm and from (8), we have$x^j_{t+1} = x^j_t - \eta_t \nabla^j F(\tilde{x}_t(j))$, where $\tilde{x}_t(j)$ is the concatenated model parameters with staleness in which $\tilde{x}^i_t(j) = x^i_{t-\tau^j(i)}$. Note that we always have $\tau^j(i) \le \tau, \forall i, j$. To help proving the proposition, we first prove a lemma.

Lemma 1.

$$< x_t - x_*, \nabla F_t(x_t) >= \frac{1}{2}\eta_t \sum_{j=1}^{m} \|\nabla^j F(\tilde{x}_t(j))\|^2 - \frac{D_{t+1} - D_t}{\eta_t}$$

$$+ \sum_{j=1}^{m} < x^j_t - x^j_*, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > . \tag{12}$$

Proof. $D_{t+1} - D_t$

$$= \frac{1}{2}\sum_{j=1}^{m} \left(\|x^j_t - \eta_t \nabla^j F(\tilde{x}_t(j)) - x^j_*\|^2 - \|x^j_t - x^j_*\|^2\right)$$

$$= \sum_{j=1}^{m} \Big(\frac{1}{2}\|\eta_t \nabla^j F(\tilde{x}_t(j))\|^2 - \eta_t < x^j_t - x^j_*, \nabla^j F(\tilde{x}_t(j)) > \Big)$$

$$= \frac{1}{2}\eta_t^2 \sum_{j=1}^{m} \|\nabla^j F(\tilde{x}_t(j))\|^2 - \eta_t < x_t - x_*, \nabla F(x_t) >$$

$$+ \eta_t \sum_{j=1}^{m} \Big\langle x^j_t - x^j_*, \nabla^j F(x_t) - \nabla^j F(\tilde{x}_t(j)) > . \tag{13}$$

Dividing the above equation by η_t, we can get the lemma. □

Another important fact for our analysis is

$$\sum_{t=a}^{b} \frac{1}{\sqrt{t}} \le \int_{a-1}^{b} \frac{1}{\sqrt{t}} dt = 2(\sqrt{b} - \sqrt{a-1}). \tag{14}$$

We now come to evaluate the regret R up to iteration T. By the definition in (10) and, we have

$$R = \frac{1}{T}\sum_{t} F_t(x_t) - F(x_*) = \frac{1}{T}\sum_{t=1}^{T} F_t(x_t) - \frac{1}{T}\sum_{t=1}^{T} F_t(x_*) \tag{15}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\Big(F_t(x_t) - F_t(x_*)\Big) \le \frac{1}{T}\sum_{t=1}^{T} < x_t - x_*, \nabla F_t(x_t) > \tag{16}$$

where (16) follows from the convexity of the loss functions. Inserting the result from lemma 1, we can get

$$\begin{aligned} T \cdot R \le & \sum_{t=1}^{T}\Big(\frac{1}{2}\eta_t \sum_{j=1}^{m} \|\nabla^j F(\tilde{x}_t(j))\|^2 - \frac{D_{t+1} - D_t}{\eta_t} \\ & + \sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > \Big) \\ = & \sum_{t=1}^{T} \frac{1}{2}\eta_t \sum_{j=1}^{m} \|\nabla^j F(\tilde{x}_t(j))\|^2 - \sum_{t=1}^{T} \frac{D_{t+1} - D_t}{\eta_t} \\ & + \sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > . \end{aligned} \tag{17}$$

We look into the three terms of (17) and bound them.

For the first term, we have

$$\sum_{t=1}^{T} \frac{1}{2}\eta_t \sum_{j=1}^{m} \|\nabla^j F(\tilde{x}_t(j))\|^2 \le \sum_{t=1}^{T} \frac{1}{2}\eta_t m G^2 = \sum_{t=1}^{T} \frac{1}{2}\frac{\eta}{\sqrt{t}} m G^2 \le \eta m G^2 \sqrt{T}. \tag{18}$$

For the second term, we have

$$-\sum_{t=1}^{T} \frac{D_{t+1} - D_t}{\eta_t} = \frac{D_1}{\eta_1} - \frac{D_{t+1}}{\eta_t} + \sum_{t=2}^{T} D_t\left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \tag{19}$$

$$\le \frac{D^2}{\eta} - 0 + \sum_{t=2}^{T} \frac{D^2}{\eta}\left(\sqrt{t} - \sqrt{t-1}\right) = \frac{D^2\sqrt{T}}{\eta}. \tag{20}$$

Finally we come to the third term. We have

$$\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) >$$

$$\le \sum_{t=1}^{T}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \cdot \|\nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j))\| \tag{21}$$

$$\le \sum_{t=1}^{T}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \cdot L_j \|x_t - \tilde{x}_t(j)\| \tag{22}$$

$$\le \sum_{t=1}^{T}\sum_{j=1}^{m} L_j \|x_t^j - x_*^j\| \cdot \sum_{i=1}^{m} \|x_t^i - x_{t-\tau^j(i)}^i\|. \tag{23}$$

If $\tau^j(i) \ge 0$, we have

$$\|x_t^i - x_{t-\tau^j(i)}^i\| = \Big\| \sum_{q=t-\tau^j(i)}^{t-1} (x_{q+1}^i - x_q^i) \Big\| \le \sum_{q=t-\tau^j(i)}^{t-1} \|\eta_q \nabla^i F_q(\tilde{x}_q(i))\|$$

$$\le \sum_{q=t-\tau^j(i)}^{t-1} \eta_q G. \tag{24}$$

If $\tau^j(i) < 0$, by similar technique, we can also get (24). Inserting (24) into (23), we have

$$\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) >$$

$$\le G m \sum_{t=1}^{T}\sum_{j=1}^{m} L_j \|x_t^j - x_*^j\| \cdot \sum_{q=t-\tau}^{t-1} \eta_q \tag{25}$$

$$\le G m L_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \sum_{j=1}^{m} \|x_t^j - x_*^j\| \tag{26}$$

$$\le G m^{\frac{3}{2}} L_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \|x_t - x_*\| \tag{27}$$

$$\le G D m^{\frac{3}{2}} L_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q. \tag{28}$$

(21) is from triangle inequality. (22) comes from the Assumption 1's blockwise Lipschitz continuity. (27) comes from the fact

$$\frac{1}{m}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \le \frac{1}{\sqrt{m}}\sqrt{\sum_{j=1}^{m} \|x_t^j - x_*^j\|^2} = \frac{1}{\sqrt{m}}\|x_t - x_*\|. \tag{29}$$

For the last parts of (28), we have

$$\sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \le \sum_{t=1}^{\tau} \eta_1 t + \sum_{t=\tau+1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \le \frac{\eta\tau(\tau+1)}{2} + \sum_{t=\tau+1}^{T} \frac{\tau\eta}{\sqrt{t-\tau}} \tag{30}$$

$$\le \frac{\eta\tau(\tau+1)}{2} + 2\tau\eta\sqrt{T-\tau} \tag{31}$$

$$= \frac{\eta\tau}{2}(\tau + 1 + 4\sqrt{T}), \tag{32}$$

where (31) is from the fact (14). Combining (28) and (32), we get

$$\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) >$$

$$\le \frac{1}{2} G D m^{\frac{3}{2}} L_{\max} \eta\tau(\tau + 1 + 4\sqrt{T}) \tag{33}$$

Combining (17), (18), (20) and (33), and dividing by T, we have

$$R \le \frac{\eta m G^2}{\sqrt{T}} + \frac{D^2}{\eta\sqrt{T}} + \frac{1}{2} G D m^{\frac{3}{2}} L_{\max} \eta\tau \frac{1}{\sqrt{T}}\left(\frac{\tau+1}{\sqrt{T}} + 4\right) = O\left(\frac{1}{\sqrt{T}}\right).$$

■