Hogwild! over Distributed Local Data Sets with Linearly Increasing Mini-Batch Sizes

[*]**Nhuong V. Nguyen** [*]**Toan N. Nguyen** [†]**Phuong Ha Nguyen**
[‡]**Quoc Tran-Dinh** [††]**Lam M. Nguyen** [*§]**Marten van Dijk**
[*]UConn [†]eBay [‡]UNC Chapel Hill [††]IBM Research [§]CWI Amsterdam

Abstract

Hogwild! implements asynchronous Stochastic Gradient Descent (SGD) where multiple threads in parallel access a common repository containing training data, perform SGD iterations, and update shared state that represents a jointly learned (global) model. We consider big data analysis where training data is distributed among local data sets in a heterogeneous way – and we wish to move SGD computations to local compute nodes where local data resides. The results of these local SGD computations are aggregated by a central "aggregator" which mimics Hogwild!. We show how local compute nodes can start choosing small mini-batch sizes which increase to larger ones in order to reduce communication cost (round interaction with the aggregator). We improve state-of-the-art literature and show $O(\sqrt{K})$ communication rounds for heterogeneous data for strongly convex problems, where K is the total number of gradient computations across all local compute nodes. For our scheme, we prove a *tight* and novel non-trivial convergence analysis for strongly convex problems for *heterogeneous* data which does not use the bounded gradient assumption as seen in many existing publications. The tightness is a consequence of our proofs for lower and upper bounds of the convergence rate, which show a constant factor difference. We show experimental results for plain convex and non-convex problems for biased (i.e., heterogeneous) and unbiased local data sets.

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

1 Introduction

The optimization problem for training many Machine Learning (ML) models using a training set $\{\xi_i\}_{i=1}^M$ of M samples can be formulated as a finite-sum minimization problem as follows

$$\min_{w\in\mathbb{R}^d}\left\{F(w)=\frac{1}{M}\sum_{i=1}^{M}f(w;\xi_i)\right\}.$$

The objective is to minimize a loss function with respect to model parameters w. This problem is known as empirical risk minimization and it covers a wide range of convex and non-convex problems from the ML domain, including, but not limited to, logistic regression, multi-kernel learning, conditional random fields and neural networks. In this paper, we are interested in solving the following more general stochastic optimization problem with respect to some distribution $\mathcal{D}$:

$$\min_{w\in\mathbb{R}^d}\left\{F(w)=\mathbb{E}_{\xi\sim\mathcal{D}}[f(w;\xi)]\right\}, \tag{1}$$

where F has a Lipschitz continuous gradient and f is bounded from below for every ξ.

Big data analysis in the form of ML over a large training set distributed over local databases requires computation to be moved to compute nodes where local training data resides. Local SGD computations are communicated to a central "aggregator" who maintains a global model. Local computations are executed in parallel and resulting SGD updates arrive out-of-order at the aggregator. For this purpose we need robust (in terms of convergence) asynchronous SGD.

Our approach is based on the Hogwild! [Recht et al., 2011] recursion

$$w_{t+1}=w_t-\eta_t\nabla f(\hat{w}_t;\xi_t), \tag{2}$$

where $\hat{w}_t$ represents the vector used in computing the gradient $\nabla f(\hat{w}_t;\xi_t)$ and whose vector entries have been read (one by one) from an aggregate of a mix of previous updates that led to w_j, $j \leq t$. In a single-thread setting where updates are done in a fully consistent way, i.e. $\hat{w}_t = w_t$, yields SGD with diminishing step sizes $\{\eta_t\}$.

Recursion (2) models asynchronous SGD. We define the amount of asynchronous behavior by function $\tau(t)$:

Definition 1. *We say that the sequence* $\{w_t\}$ *is* consistent with a delay function τ *if, for all* t, *vector* $\hat{w}_t$ *includes the aggregate of the updates up to and including those made during the* $(t-\tau(t))$*-th iteration*[*], *i.e.,* $\hat{w}_t = w_0 - \sum_{j\in\mathcal{U}} \eta_j \nabla f(\hat{w}_j;\xi_j)$ *for some* $\mathcal{U}$ *with* $\{0,1,\ldots,t-\tau(t)-1\} \subseteq \mathcal{U}$.

Our main insight is that the asynchronous SGD framework based on Hogwild! can resist much larger delays than the natural delays caused by the network communication infrastructure, in fact, it turns out that $\tau(t)$ can scale as much as $\approx \sqrt{t/\ln t}$ for strongly convex problems [Nguyen et al., 2018, 2019a]. This means that recurrence (2) can be used/exploited in an asynchronous SGD implementation over distributed local data sets where much more *asynchronous behavior is introduced by design.*

In our setting where SGD recursions are executed locally at compute nodes where biased (i.e., heterogeneous) local training data resides, compute nodes execute SGD recursions in 'rounds'. From the perspective of a local compute node, a round consists of the sequence of SGD recursions between two consecutive 'update' communications to the central aggregator. Within a round, w_t is iteratively updated by subtracting $\eta_t \nabla f(\hat{w}_t;\xi_t)$. The sum of updates $\sum_{t\in round} \eta_t \nabla f(\hat{w}_t;\xi_t)$ is communicated to the aggregator at the end of the round after which a new round starts leading to a next sum of updates to be communicated at the end of the next round. Locally, compute nodes receive (out of sync) broadcast messages with the current global model from the central aggregator (according to some strategy, e.g., on average one or two such messages per round). This is used to replace their local model w_t computed so far. Details are given in Section 3.

Rather than having each round execute the same/constant number of SGD recursions, this paper builds on our main observation that we are allowed to introduce asynchronous behavior by design while still maintaining convergence (as we will see at a rate that matches a *lower bound* up to a constant). We propose to *increase the number of SGD iterations performed locally at each compute node from round to round.* This *reduces the amount of network communication* compared to the straightforward usage of recurrence (2) where each compute node performs a fixed number of SGD iterations within each round.

In the distributed data setting, each local compute node may only execute for a fraction of an epoch (where the number of iterations, i.e., gradient computations, in an epoch is equal to the size of the global big training data set defined as the collection of all local training data sets together). In order to disperse to all local compute nodes information from local updates by means of updating the global model and receiving feedback from the global model (maintained at the central aggregator), there needs to be a sufficient number of round interactions. For the best convergence, we need to have more round interactions at the very start where local updates contain the most (directional) information about (where to find) the global minimum to which we wish to converge. This corresponds to small 'sample' sizes (measured in the number of local SGD updates within a round) in the beginning. And in order to gain as much useful information about where to find the global minimum, initial local updates should use larger step sizes (learning rate).

From our theory we see that in order to bootstrap convergence it is indeed the best to start with larger step sizes and start with rounds of small sample size after which these rounds should start increasing in sample size and should start using smaller and smaller step sizes for best performance (in terms of minimizing communication while still achieving high test accuracy). Experiments confirm our expectations.

Contributions. For the distributed data setting where SGD recursions are executed locally at compute nodes where local training data resides and which update a global model maintained at a centralized aggregator, we introduce a new SGD algorithm based on Hogwild! [Recht et al., 2011] which does not use fixed-sized min-batch SGD at the local compute nodes but uses increasing mini-batch (sample) sizes from round interaction to round interaction:

[I] The compute nodes and server can work together to create a global model in asynchronous fashion, where we assume that messages/packets never drop; they will be re-sent but can arrive out of order. In Theorem 1 we characterize distribution $\mathcal{D}$ in the stochastic optimization problem (1) to which the global model relates.

[II] Given a specific (strongly convex, plain convex or non-convex) stochastic optimization problem, we may assume (believe in) a delay function τ which characterizes the maximum asynchronous behavior which our algorithm can resist for the specific problem. Given the delay function τ, we provide a general recipe for constructing increasing sample size sequences and diminishing round step size sequences so that our algorithm

[*](2) defines the $(t+1)$-th iteration, where $\eta_t \nabla f(\hat{w}_t;\xi_t)$ represents the $(t+1)$-th update.

maintains τ as an invariant. For strongly convex problems, [Nguyen et al., 2018, 2019a] prove that $\tau(t)$ can be as large as $\approx \sqrt{t/\ln t}$ for which our recipe shows a diminishing round step size sequence of $O(\frac{\ln i}{i^2})$, where i indicates the round number, that allows a sample size sequence of $\Theta(\frac{i}{\ln i})$; the sample size sequence can almost *linearly* increases from round to round.

[III] For strongly convex problems with 'linearly' increasing sample size sequences $\Theta(\frac{i}{\ln i})$ we prove in Theorem 2 an upper bound of $O(1/t)$ on the expected convergence rate $\mathbb{E}[\|w_t - w_*\|^2]$, where w_* represents the global minimum in (1) and t is the SGD iteration number (each local node computes a subset of the w_t). In fact, the concrete expression of the upper bound attains for increasing t the best possible convergence rate (among stochastic first order algorithms) within a constant factor $\leq 8 \cdot 36^2$, see Corollary 1 which directly applies the lower bound from [Nguyen et al., 2019b].

[IV] Let K be the total number of gradient computations (summed over all local nodes) needed for the desired test accuracy, and let T be the number of communication rounds in our algorithm. Then, T scales less than linear with K due to the increasing sample size sequence (if a constant sample size sequence is used, i.e., fixed-sized mini-batch SGD at each of the local compute nodes, then $T = O(K)$). For strongly convex problems with diminishing step sizes we show $T = O(\sqrt{K})$ for heterogeneous local data while having $O(1/K)$ convergence rate. This implies that using diminishing step sizes give a much better performance compared to constant step sizes in terms of communication.

[V] Experiments for linearly increasing sample size sequences for strongly convex problems as well as plain convex and non-convex problems confirm the robustness of our algorithm in terms of good test accuracies (our theoretical understanding from strongly convex problems seems to generalize to plain and non-convex problems). We use biased local training data sets (meaning different compute nodes see differently biased subsets of training data) and compare to unbiased local training data sets.

2 Related Work

Unbiased Local Data (iid). For strongly convex problems with unbiased local data sets, [Stich, 2018] showed $O(1/K)$ convergence rate for $O(\sqrt{K})$ communication rounds. For the iid case this was improved by [Spiridonoff et al., 2020] to just 1 communication round, where each client performs local SGD separately after which in "one shot" all local models are aggregated (averaged) – this corresponds to $O(n)$ total communication for n clients. This result was generalized by [Khaled et al., 2020]. Based on the strong Polyak-Lojasiewicz (PL) assumption (which is a generalization of strong convexity but covers certain nonconvex models), [Yu and Jin, 2019] proved for the iid case a convergence rate of $O(1/K)$ with $\log(K/n)$ communication rounds with an exponentially increasing sample size sequence. For non-convex problems, [Yu and Jin, 2019] proved for the iid case the standard convergence rate $O(1/\sqrt{K})$ (as defined for non-convex problems) with $O(\sqrt{K}\log(K/n^2))$ communication rounds.

Biased Local Data (heterogeneous). Our focus is on heterogeneous data between different clients (this needs more communication rounds in order to achieve convergence, one-shot averaging is not enough): [Khaled et al., 2020] were the first to analyze the convergence rate for plain convex problems in this scenario. They use the bounded variance assumption in their analysis with constant step-size and with sample size sequences where sample sizes are bound by an a-priori set parameter H. They prove that $O(1/\sqrt{K})$ convergence rate (optimal for plain convex) is achieved for $O(K^{3/4})$ communication rounds (see their Corollary 5 and notice that their algorithm uses $(K/n)/H$ communication rounds). For strongly convex problems in the *heterogeneous* case (without assuming bounded variance), we show that *convergence rate $O(1/K)$ (optimal for strongly convex) is achieved for $O(\sqrt{K})$ communication rounds.*

Asynchronous Training. Asynchronous training [Zinkevich et al., 2009, Lian et al., 2015, Zheng et al., 2017, Meng et al., 2017, Stich, 2018, Shi et al., 2019] is widely used in traditional distributed SGD. Hogwild!, one of the most famous asynchronous SGD algorithms, was introduced in [Recht et al., 2011] and various variants with a fixed and diminishing step size sequences were introduced in [Mania et al., 2015, De Sa et al., 2015, Leblond et al., 2018, Nguyen et al., 2018]. Typically, asynchronous training converges faster than synchronous training in real time due to parallelism. This is because in a synchronized solution compute nodes have to wait for the slower ones to communicate their updates after which a new global model can be downloaded by everyone. This causes high idle waiting times at compute nodes. Asynchronous training allows compute nodes to continue executing SGD recursions based on stale global models. For non-convex problems, synchronous training [Saeed Ghadimi, 2013] and asynchronous training with bounded staleness [Lian et al., 2015], or in our terminology bounded delay, achieves the same convergence rate of $O(1/\sqrt{K})$, where K is the total number of gradient computations.

The methods cited above generally use mini-batch SGD (possibly with diminishing step sizes from round to round) at each of the distributed computing threads, hence, parallelism will then lead to asynchronous behavior dictated by a delay τ which can be assumed to be bounded. Assuming bounded delays, the convergence rate is mathematically analysed in the papers cited above with the exception of [Nguyen et al., 2018, 2019a] which also analyses the convergence rate for unbounded delays (i.e., increasing delay functions τ).

As explained in this introduction, *this paper exploits the advantage of being able to resist much more asynchronous behavior than bounded delay.* We show how one can use diminishing step sizes alongside increasing sample sizes (mini-batch sizes) from round to round. This provides a technique complimentary to [Zinkevich et al., 2009, Lian et al., 2015, 2017, Zheng et al., 2017, Meng et al., 2017, Stich, 2018, Shi et al., 2019] with which current asynchronous training methods can be enhanced at the benefit of reduced communication – which is important when training over distributed local data sets in big data analysis. For example, rather than sending gradients to the server after each local update, which is not practical for edge devices (such as mobile or IoT devices) due to unreliable and slow communication, [Shi et al., 2019] introduces the idea of using a tree like communication structure which aggregates local updates in pairs from leafs to root – this technique for meeting throughput requirements at the centralized server can be added to our technique of increasing sample sizes from round to round. As another example, [Lian et al., 2017] introduces asynchronous decentralized SGD where local compute nodes do not communicate through a centralized aggregator but instead perform a consensus protocol – our technique of increasing sample sizes is complementary and can possibly be beneficial to use in this decentralized network setting. Similarly, our technique may apply to [Jie Xu, 2020], where the authors studied a new asynchronous decentralized SGD with the goal of offering privacy guarantees. We stress that our setting is asynchronous centralized SGD and is completely different from asynchronous decentralized SGD algorithms as in [Shi et al., 2019, Lian et al., 2017, Jie Xu, 2020].

Federated Learning and Local SGD. Federated Learning (FL) [Chen et al., 2016, McMahan et al., 2016] is a distributed machine learning approach which enables training on a large corpus of decentralized data located on devices like mobile phones or IoT devices. Federated learning brings the concept of "bringing the code to the data, instead of the data to the code" [Bonawitz et al., 2019]. Google [Konečný et al., 2016] demonstrated FL for the first time at a large scale when they conducted experiments of training a global model across all mobile devices via the Google Keyboard Android application [McMahan and Ramage, 2017].

Original FL requires synchrony between the server and clients (compute nodes). It requires that each client sends a full model back to the server in each round and each client needs to wait for the next computation round. For large and complicated models, this becomes a main bottleneck due to the asymmetric property of internet connection and the different computation power of devices [Chen et al., 2019a, Konečný et al., 2016, Wang et al., 2019, Hsieh et al., 2017].

In [Xie et al., 2019, Chen et al., 2019b] asynchronous training combined with federated optimization is proposed. Specifically, the server and workers (compute nodes) conduct updates asynchronously: the server immediately updates the global model whenever it receives a local model from clients. Therefore, the communication between the server and workers is non-blocking and more effective. We notice that [Xie et al., 2019] provides a convergence analysis, while [Chen et al., 2019b] does not.

In [Li et al., 2019], the authors introduce FedProx which is a modification of FedAvg (i.e., original FL algorithm of [McMahan et al., 2016]). In FedProx, the clients solve a proximal minimization problem rather than traditional minimization as in FedAvg. For theory, the authors use B-local dissimilarity and bounded dissimilarity assumptions for the global objective function. This implies that there is a bounded gradient assumption applied to the global objective function. Moreover, their proof requires the global objective function to be strongly convex.

One major shortcoming in the terms of convergence analysis of asynchronous SGD in many existing publications is that the bounded gradients and strongly convex assumptions are used together, e.g. [Lian et al., 2015, 2017, Mania et al., 2015, De Sa et al., 2015, Jie Xu, 2020, Xie et al., 2019, McMahan et al., 2016, Li et al., 2019]. However, the bounded gradient assumption is in conflict with assuming strong convexity as explained in [Nguyen et al., 2018, 2019a]. It implies that the convergence analysis should not use these two assumptions together to make the analysis complete.

Our method of increasing sample sizes together with its convergence analysis for strongly objective functions (which does not use the bounded gradient assumption) complements the related work in FL and local SGD. This paper analyses and demonstrates the promise of this new technique but does not claim a full end-to-end implementation of FL or distributed SGD with asynchronous learning.

3 Hogwild! & Increasing Sample Sizes

The next subsections explain our proposed algorithm together with how to set parameters in terms of a concrete round to round diminishing step size sequence and increasing sample size sequence.

3.1 Asynchronous SGD over Local Data Sets

Algorithm 1 ComputeNode$_c$ – Local Model

```
1:  procedure SETUP($n$): Initialize increasing sample size sequences $\{s_i\}_{i\geq 0}$ and $\{s_{i,c} \approx p_c s_i\}_{i\geq 0}$ for each compute node $c \in \{1,\ldots,n\}$, where $p_c$ scales the importance of compute node $c$. Initialize diminishing round step sizes $\{\bar{\eta}_i\}_{i\geq 0}$, a permissible delay function $\tau(\cdot)$ with $t-\tau(t)$ increasing in $t$, and a default global model for each compute node to start with.
2:  end procedure
3:
4:  procedure ISRRECEIVE($\hat{v}, k$): This Interrupt Service Routine is called whenever a broadcast message with a new global model $\hat{v}$ is received from the server. Once received, the compute node's local model $\hat{w}$ is replaced with $\hat{v}$ from which the latest accumulated update $\bar{\eta}_i \cdot U$ of the compute node in the ongoing round (as maintained in line 17 below) is subtracted. (We notice that the $k$th broadcast message containing a global model $\hat{v}$ from the server is transmitted by the server as soon as the updates up to and including rounds $0,\ldots,k-1$ from *all* compute nodes have been received; thus $\hat{v}$ includes all the updates up to and including round $k-1$.)
5:  end procedure
6:
7:  procedure MAINCOMPUTENODE($\mathcal{D}_c$)
8:      $i = 0$, $\hat{w} = \hat{w}_{c,0,0}$
9:      while True do
10:         $h = 0$, $U = 0$
11:         while $h < s_{i,c}$ do
12:             $t_{glob} = s_0 + \ldots + s_i - (s_{i,c} - h) - 1$
13:             $t_{delay} = s_k + \ldots + s_i - (s_{i,c} - h)$
14:             while $\tau(t_{glob}) < t_{delay}$ do nothing
15:             Sample uniformly at random $\xi$ from $\mathcal{D}_c$
16:             $g = \nabla f(\hat{w}, \xi)$
17:             $U = U + g$
18:             Update model $w = \hat{w} - \bar{\eta}_i \cdot g$
19:                                 ▷ $w$ represents $w_{c,i,h+1}$
20:             Update model $\hat{w} = w$
21:             $h$++
22:         end while
23:         Send $(i, c, U)$ to the Server.
24:         $i$++
25:     end while
26: end procedure
```

Compute node $c \in \{1,\ldots,n\}$ updates its local model $\hat{w}$ according to Algorithm 1. Lines 15, 16, 18, and 20 represent an SGD recursion where ξ is sampled from distribution $\mathcal{D}_c$, which represents c's local data set. Variable U, see line 17, keeps track of the sum of the gradients that correspond to $s_{i,c}$ samples during c's i-th local round. This information is send to the server in line 23, who will multiply U by the round step size $\bar{\eta}_i$ and subtract the result from the global model $\hat{v}$. In this way each compute node contributes updates U which are aggregated at the server. As soon as the server has aggregated each compute node's updates for their first k local rounds, the server broadcasts global model $\hat{v}$. As soon as c receives $\hat{v}$ it replaces its local model $\hat{w}$ with $\hat{v} - \bar{\eta}_i \cdot U$ (this allows the last computed gradients in U that have not yet been aggregated into the global model at the server not to go to waste).

Line 14 shows that a compute node c will wait until t_{delay} becomes smaller than $\tau(t_{glob})$. This will happen as a result of ISRRECEIVE($\hat{v}, k$) receiving broadcast message $\hat{v}$ together with a larger k after which the ISR computes a new (smaller) t_{delay}. Line 14 guarantees the invariant $t_{delay} \leq \tau(t_{glob})$, where τ is initialized to some "permissible" delay function which characterizes the amount of asynchronous behavior we assume the overall algorithm can tolerate (in that the algorithm has fast convergence leading to good test accuracy).

Supplemental Material A has all the detailed pseudo code with annotated invariants.

We want to label the SGD recursions computed in each of the MAINCOMPUTENODE($\mathcal{D}_c$) applications for $c \in \{1,\ldots,n\}$ to an iteration count t that corresponds to recursion (2): We want to put all SGD recursions computed by each compute node in sequential order such that it is as if we used (2) on a single machine. This will allow us to analyse whether our algorithm leads to a sequence $\{w_t\}$ which is consistent with the initialized permissible delay function τ. In order to find an ordering based on t we define in Supplemental Material B.1 a mapping ρ from the annotated labels (c,i,h) in MAINCOMPUTENODE to t and use this to prove the following theorem:

Theorem 1. *Our setup, compute node, and server algorithms produce a sequence $\{w_t\}$ according to recursion (2) where $\{\xi_t\}$ are selected from distribution $\mathcal{D} = \sum_{c=1}^n p_c \mathcal{D}_c$. Sequence $\{w_t\}$ is consistent with delay function τ as defined in* SETUP.

The theorem tells us that the algorithms implement recursion (2) for distribution $\mathcal{D} = \sum_{c=1}^n p_c \mathcal{D}_c$, i.e., a convex combination of each of the (possibly biased) local distributions (data sets). Scaling factors p_c represent a distribution (i.e., they sum to 1) and are used to compute local sample sizes $s_{i,c} \approx p_c s_i$, where $s_i = \sum_{c=1}^n s_{i,c}$ indicates the total number of samples in rounds i across all compute nodes.

We remark that our asynchronous distributed SGD is compatible with the more general recursion mentioned in [Nguyen et al., 2018, 2019a] and explained in Supplemental Material C.1. In this recursion each client can apply a "mask" which indicates the entries of the local model that will be considered. This allows each client to only transmit the local model entries corresponding to the mask.

3.2 Delay τ

We assume that messages/packets never drop. They will be resent but can arrive out-of-order. We are robust against this kind of asynchronous behavior: The amount of asynchronous behavior is limited by $\tau(\cdot)$; when the delay is getting too large, then the client (local compute node) enters a wait loop which terminates only when ISRReceive receives a more recent global model $\hat{v}$ with higher k (making t_{delay} smaller). Since $\tau(t)$ increases in t and is much larger than the delays caused by network latency and retransmission of dropped packets, asynchronous behavior due to such effects will not cause clients to get stuck in a waiting loop. We assume different clients have approximately the same speed of computation which implies that this will not cause fast clients having to wait for long bursts of time.[†]

We exploit the algorithm's resistance against delays $\tau(t)$ by using increasing sample size sequences $\{s_{i,c}\}$. Since the server only broadcasts when all clients have communicated their updates for a "round" k, increasing sample sizes implies that t_{delay} can get closer to $\tau(t_{glob})$. So, sample size sequences should not increase too much: We require the property that there exists a threshold d such that for all $i \geq d$,

$$\tau\left(\sum_{j=0}^{i} s_j\right) \geq 1 + \sum_{j=i-d}^{i} s_j. \tag{3}$$

In Supplemental Material B.2 we show that this allows us to replace condition $\tau(t_{glob}) < t_{delay}$ of the waiting loop by $i > k + d$ when $i \geq d$ while still guaranteeing $t_{delay} \leq \tau(t_{glob})$ as an invariant. In practice, since sample sizes increase, we only need to require (3) for $d = 1$ (which means we allow a local lag of one communication round) in order to resist asynchronous behavior due to network latency.

[†]When entering a waiting loop, the client's operating system should context switch out and resume other computation. If a client c is an outlier with slow computation speed, then we can adjust p_c to be smaller in order to have its mini-batch/sample size $s_{i,c}$ be proportionally smaller; this will change distribution $\mathcal{D}$ and therefore change the objective function of (1).

3.3 Recipe Sample Size Sequence

Given a fixed budget/number of gradient computations K which the compute nodes together need to perform, an increasing sample size sequence $\{s_i\}$ reduces the number T of communication rounds/interactions (defined by update messages coming from the compute nodes with broadcast messages from the server). Convergence to an accurate solution must be guaranteed, that is, $\{s_i\}$ has to satisfy (3) if we assume τ is indeed a permissible delay function.

Supplemental Material B.3 proves how a general formula for function τ translates into an increasing sample size sequence $\{s_i\}$ that satisfies (3).

Lemma 1. *Let $g > 1$. Suppose that $\tau(x) = M_1 + (x + M_0)^{1/g}$ for some $M_1 \geq d + 2$ and $M_0 \geq ((m+1)(g-1)/g)^{g/(g-1)}$, where $m \geq 0$ is an integer. Then*

$$s_i = \left\lceil \frac{1}{d+1}\left(\frac{m+i+1}{d+1}\frac{g-1}{g}\right)^{1/(g-1)} \right\rceil$$

satisfies property (3).

The above lemma is a direct consequence of Supplemental Material B.3 which has a more general proof that also allows functions such as $\tau(x) = M_1 + ((x + M_0)/\ln(x + M_0))^{1/g}$ (needed for analysing the convergence of strongly convex problems with $g = 2$).

3.4 Recipe Round Step Size Sequence

As soon as we have selected an increasing sample size sequence based on τ, Supplemental Material B.4 shows how we can translate the diminishing step size sequence $\{\eta_t\}$ of recurrence (2) to a diminishing round step size sequence $\{\bar{\eta}_i\}$ that only diminishes with every mini-batch s_i from round to round. The lemma below is a direct consequence of Supplemental Material B.4 which has a slightly more general statement.

Lemma 2. *Let $0 \leq q \leq 1$ and $\{E_t\}$ a constant or increasing sequence with $E_t \geq 1$. For q and $\{E_t\}$ consider the set $\mathcal{Z}$ of diminishing step size sequences $\{\eta_t\}$ in recurrence (2) with $\eta_t = \alpha_t/(\mu(t+E_t)^q)$ where $\{\alpha_t\}$ is some sequence of values with $\alpha_0 \leq \alpha_t \leq 3 \cdot \alpha_0$.*

We assume sample size sequence $\{s_i\}$ of Lemma 1 for $g \geq 2$. For $i \geq 0$, we define $\bar{E}_i = E_{\sum_{j=0}^{i} s_j}$. We define $\bar{E}_{-1} = E_0$. If $\bar{E}_i \leq 2\bar{E}_{i-1}$ for $i \geq 0$ and if $s_0 - 1 \leq E_0$, then there exists a diminishing step size sequence $\{\eta_t\}$ in set $\mathcal{Z}$ such that

$$\eta_t = \frac{\alpha_t}{\mu(t+E_t)^q} = \frac{\alpha_0}{\mu((\sum_{j=0}^{i-1} s_j) + \bar{E}_{i-1})^q} \overset{\text{DEF}}{=} \bar{\eta}_i$$

for $t \in \{(\sum_{j=0}^{i-1} s_j), \ldots, (\sum_{j=0}^{i-1} s_j) + s_i - 1\}$.

Notice $s_i = \Theta(i^{1/(g-1)})$ and $\bar{\eta}_i = O(i^{-q\cdot(1+1/(g-1))})$.

In Supplemental Material C.3 we discuss plain and non-convex problems. We argue in both cases to choose a diminishing step size sequence of $O(t^{-1/2})$, i.e., $q = 1/2$, and to experiment with different increasing sample size sequences $\Theta(i^{1/(g-1)})$, for $g \geq 2$, to determine into what extent the presented asynchronous SGD is robust against delays. Substituting $p = 1/(g-1) \in (0,1]$ gives sample size sequence $\Theta(i^p)$ and round step size sequence $O(i^{-(1+p)/2})$. It turns out that $p = 1$ gives a good performance and this confirms our intuition that the results from our theory on strongly convex functions in the next section generalizes in that also plain and non-convex problems have fast convergence for linearly increasing sample size sequences.

4 Convergence Rate for Strongly Convex problems

In this section we provide a round step size sequence and a sample size sequence for strongly convex problems. We show tight upper and lower bounds. For strongly convex problems, we assume the following:

Assumption 1 (L-smooth)**.** *$f(w;\xi)$ is L-smooth for every realization of ξ, i.e., there exists a constant $L > 0$ such that, $\forall w, w' \in \mathbb{R}^d$,*

$$\|\nabla f(w;\xi) - \nabla f(w';\xi)\| \leq L\|w - w'\|.$$

Assumption 2. *$f(w;\xi)$ is convex for every realization of ξ, i.e., $\forall w, w' \in \mathbb{R}^d$,*

$$f(w;\xi) - f(w';\xi) \geq \langle \nabla f(w';\xi), (w - w')\rangle.$$

Assumption 3 (μ-strongly convex)**.** *The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is a μ-strongly convex, i.e., there exists a constant $\mu > 0$ such that $\forall w, w' \in \mathbb{R}^d$,*

$$F(w) - F(w') \geq \langle \nabla F(w'), (w - w')\rangle + \frac{\mu}{2}\|w - w'\|^2.$$

Being strongly convex implies that F has a global minimum w_*. For w_* we assume:

Assumption 4 (Finite σ)**.** *Let $N = 2\mathbb{E}[\|\nabla f(w_*;\xi)\|^2]$ where $w_* = \arg\min_w F(w)$. We require $N < \infty$.*

In this section we let f be L-smooth, convex, and let the objective function $F(w) = \mathbb{E}_{\xi\sim\mathcal{D}}[f(w;\xi)]$ be μ-strongly convex with finite $N = 2\mathbb{E}[\|\nabla f(w_*;\xi)\|^2]$ where $w_* = \arg\min_w F(w)$. Notice that we do not assume the bounded gradient assumption which assumes $\mathbb{E}[\|\nabla f(w;\xi)\|^2]$ is bounded for all $w \in \mathbb{R}^d$ (not only $w = w_*$ as in Assumption 4) and is in conflict with assuming strong convexity as explained in [Nguyen et al., 2018, 2019a].

4.1 Sample Size and Round Step Size Sequences

After sequentially ordering the SGD recursions from all compute nodes we end up with a Hogwild! execution as defined by recursion (2) and we may apply the results from [Nguyen et al., 2018, 2019a] that state that $\{w_t\}$ is consistent with any delay function $\tau(t) \leq \sqrt{(t/\ln t)\cdot(1 - 1/\ln t)}$. By suitably choosing such a function τ (see Supplemental Material C.2.2 for details), application of the more general Lemma 1 from Supplemental Material B.3 gives sample size sequence

$$s_i = \left\lceil \frac{m+i+1}{16(d+1)^2}\frac{1}{\ln(\frac{m+i+1}{2(d+1)})}\right\rceil = \Theta\left(\frac{i}{\ln i}\right).$$

For a fixed number of gradient computations K, the number T of communication rounds satisfies $K = \sum_{j=0}^{T} s_j$. When forgetting the $\ln i$ component, this makes T proportional to $\sqrt{K}$ – rather than proportional to K for a constant sample size sequence. This reduction in communication rounds and overall network communication is possible because we use a diminishing step size sequence (which allows us to still prove tight upper and lower bounds on the convergence rate).

For our choice of τ, we are restricted in the more general Lemma 2 from Supplemental Material B.4 to a family $\mathcal{Z}$ of step size functions with $a_0 = 12$ and $E_t = 2\tau(t)$. For strongly convex problems we may choose $q = 1$ which gives a step size sequence $\eta_t = \alpha_t/(\mu(t+E_t)^q)$ for which the convergence rate $\mathbb{E}[\|w_t - w_*\|^2] = O(1/t)$. This results in a $O\left(\frac{\ln i}{i^2}\right)$ round step size sequence

$$\bar{\eta}_i = \frac{12}{\mu}\cdot\frac{1}{\sum_{j=0}^{i-1} s_j + 2M_1 + \sqrt{\frac{(m+1)^2/4 + \sum_{j=0}^{i-1} s_j}{\ln((m+1)^2/4 + \sum_{j=0}^{i-1} s_j)}}},$$

where

$$M_1 = \max\left\{d+2, 72\cdot\frac{L}{\mu}, \frac{1}{2}\left\lceil\frac{m+1}{16(d+1)^2}\frac{1}{\ln(\frac{m+1}{2(d+1)})}\right\rceil\right\}$$

4.2 Upper Bound Convergence Rate

Based on the sequences $\{s_i\}$ and $\{\bar{\eta}_i\}$ we prove in Supplemental Material C.2.2 the following upper bound on the convergence rate for strongly convex problems:[‡]

Theorem 2. *For sample size sequence $\{s_i\}$ and round step size sequence $\{\bar{\eta}_i\}$ we have expected convergence*

[‡]We remark that this theorem also holds for our algorithm where the compute nodes compute mini-batch SGD for the sample set and it can be adapted for the more general recursion with masks as explained in Supplemental Material C.1 (with "$D > 1$").

rate

$$\mathbb{E}[\|w_t - w_*\|^2] \leq \frac{4 \cdot 36^2 \cdot N}{\mu^2}\frac{1}{t} + O\left(\frac{1}{t \ln t}\right). \quad (4)$$

where t represents the total number of gradient evaluations over all compute nodes performed so far.[§]

Notice that t is equal to n times the average number of grad evaluations $\bar{t}$ per compute node, hence, convergence rate $O(1/t) = O(1/(n\bar{t}))$ showing the expected $1/n$ dependence. We also remind the reader that $s_i = \sum_c s_{i,c}$ where $s_{i,c} \approx p_c s_i$.

We do not know whether the theory for delay functions τ for strongly convex problems gives a tight bound in terms of the maximum delay[¶] for which we can prove a tight upper bound on the convergence rate. For this reason we also experiment with the larger linear $s_i = \Theta(i)$ in the strongly convex case. (For $s_i = \Theta(i)$ we have $\bar{\eta}_i = O(i^{-2})$.)

As one benchmark we compare to using a constant step size $\eta = \bar{\eta}_i$. Supplemental Material C.2.1 analyses this case and shows how to choose the constant sample size $s = s_i$ (as large as $\frac{a}{L\mu(d+1)}$ for a well defined constant a) in order to achieve the best convergence rate.

4.3 Lower Bound Convergence Rate

Applying the lower bound from [Nguyen et al., 2019b] for first order stochastic algorithms shows the following corollary, see Supplemental Material C.2.3 for a detailed discussion (also on how fast the $O((\ln t)/t)$ term disappears and how this can be influenced by using different less increasing sample size sequences).

Corollary 1. *Among first order stochastic algorithms, upper bound (4) converges for increasing t to within a constant factor $8 \cdot 36^2$ of the (theoretically) best attainable expected convergence rate, which is at least $\frac{N}{2\mu^2}\frac{1}{t}(1 - O(\frac{\ln t}{t}))$ (for each t).*

Notice that the factor is independent of any parameters like L, μ, sparsity, or dimension of the model.

The corollary shows that non-parallel (and therefore synchronous) SGD can at most achieve a factor $8 \cdot 36^2$ faster convergence rate compared to our asynchronous SGD over (heterogeneous) local data sets. It remains an open problem to investigate second (and higher) order stochastic algorithms and whether their distributed versions attain tight convergence rates when increasing sample sizes and diminishing step sizes from round to round.

[§]Mapping ρ maps annotated labels to t: $w_t = w_{\rho(c,i,h)}$.

[¶]Permissible delay functions can possibly be larger than $\sqrt{(t/\ln t) \cdot (1 - 1/\ln t)}$.

5 Experiments

We summarize experimental results for strongly convex, plain convex and non-convex problems with *linear increasing sample sequences* and *biased versus unbiased local data sets.*

As the plain convex objective function we use logistic regression: The weight vector $\bar{w}$ and bias value b of the logistic function can be learned by minimizing the log-likelihood function J:

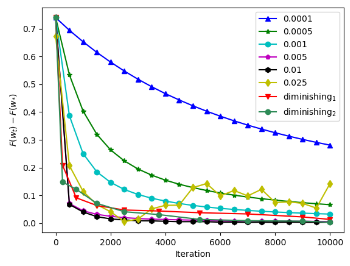
$$J = -\sum_i^M [y_i \cdot \log(\sigma_i) + (1 - y_i) \cdot \log(1 - \sigma_i)],$$

where M is the number of training samples (x_i, y_i) with $y_i \in \{0, 1\}$, and $\sigma_i = \sigma(\bar{w}, b, x_i)$ is the sigmoid function $\sigma(\bar{w}, x, b) = \frac{1}{1+e^{-(\bar{w}^T x + b)}}$. The goal is to learn a vector w^* which represents a pair $w = (\bar{w}, b)$ that minimizes J. Function J changes into a strongly convex problem by adding ridge regularization with a regularization parameter $\lambda > 0$. i.e., we minimize $\hat{J} = J + \frac{\lambda}{2}\|w\|^2$ instead of J. For simulating non-convex problems, we choose a simple neural network (LeNet) [LeCun et al., 1998] for image classification.

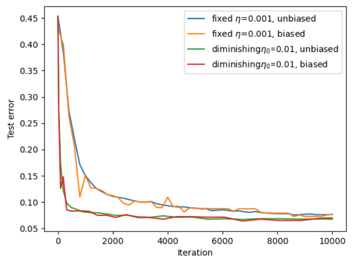
We use a linearly increasing sample size sequence $s_i = a \cdot i^c + b$, where $c = 1$ and $a, b \geq 0$. For simplicity, we choose a diminishing round step size sequence corresponding to $\frac{\eta_0}{1+\beta \cdot t}$ for the strongly convex problem and $\frac{\eta_0}{1+\beta \cdot \sqrt{t}}$ for both the plain convex and non-convex problems, where η_0 is an initial step size. The asynchronous SGD simulation is conducted with $d = 1$, see (3).

Our asynchronous SGD with linear increasing sample sizes: Figures 1a, 2a and 3a show our proposed asynchronous SGD with linear increasing sample size sequence for constant step sizes and diminishing step sizes. We conclude that diminishing step sizes achieve (approximately) a convergence which is as fast as the convergence of the best constant step size sequence. See Supplemental Material D.2.3 for details and larger sized graphs (also for other data sets); we also show experiments in D.2.1 and D.2.2 that compare using slower increasing sample size sequences with linear sample size sequences and they all achieve approximately the same convergence rate. We conclude that using a linear sample size sequence over a constant sized one does not degrade performance – on the contrary, the number of communication rounds reduces significantly. This confirms the intuition generated by our theoretical analysis for strong convex problems which generalizes to plain and non-convex problems.

Our asynchronous SGD with biased data sets: The goal of this experiment is to show that our asynchronous SGD can work well with biased (non-iid) local

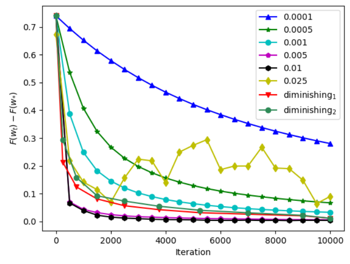


(a) Convergence rate

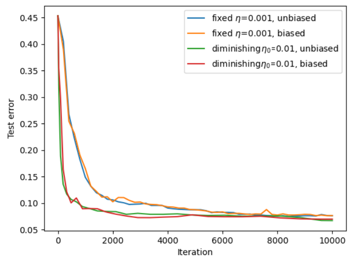


(b) Test error

Figure 1: Our asynchronous SGD for strongly convex problems: (a) The Phishing data set - various step size sequences. (b) MNIST - biased and unbiased data sets.



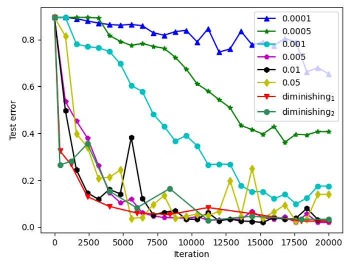
(a) Convergence rate



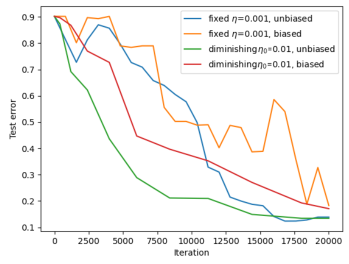
(b) Test error

Figure 2: Our asynchronous SGD for plain convex problems: (a) The Phishing data set - various step size sequences. (b) MNIST - biased and unbiased data set.

data sets meaning that different compute nodes use different distributions $\mathcal{D}_c$ (contrary to all nodes using unbiased data sets such that they all have the same $\mathcal{D}_c = \mathcal{D}$ distribution). We continue with the setting as mentioned above with an adapted initial step size η_0, see Supplemental Material D.2.4 for details. Figures 1b and 2b show no significant difference between using biased or unbiased data sets for strongly convex and plain convex problems. For the non-convex problem, Figure 3b shows that although the accuracy might fluctuate during the training process, our asynchronous SGD still achieves good accuracy in general. We conclude that our asynchronous SGD tolerates the effect of biased data sets, which is quite common in practice.



(a) Test error



(b) Test error

Figure 3: Our asynchronous SGD for non-convex problems: (a) The MNIST data set - various step size sequences. (b) MNIST - biased and unbiased data set.

Scalability: Supplemental Material D.2.5 shows that our asynchronous SGD with linear sample size sequence scales to larger number of compute nodes. The accuracy for the same total number of gradient computations stays approximately the same and the overall execution time will reach a lower limit (where increased parallelism does not help). The linear sample size sequence gives a reduced number of communication rounds (also for an increased number of compute nodes).

6 Conclusion

We provided a tight theoretical analysis for strongly convex problems over heterogeneous local data for our asynchronous SGD with increasing sample size sequences. Experiments confirm that not only strongly convex but also plain and non-convex problems can tolerate linear increasing sample sizes – this reduces the number of communication rounds.

References

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Jerry Chee and Panos Toulis. Convergence diagnostics for stochastic gradient descent with constant learning rate. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, pages 1476–1485, 2018.

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *ICLR Workshop Track*, 2016.

Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation. *arXiv preprint*, 2019a. URL `https://arxiv.org/pdf/1903.07424.pdf`.

Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation. *arXiv preprint*, 2019b. URL `https://arxiv.org/pdf/1903.07424.pdf`.

Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *NIPS*, pages 2674–2682, 2015.

Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).*, 2017.

Fei Wang Jie Xu, Wei Zhang. Asynchronous decentralized parallel stochastic gradient descent with differential privacy. *arXiv preprint arXiv:2008.09246*, 2020.

Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Tighter theory for local sgd on identical and heterogeneous data. In *International Conference on Artificial Intelligence and Statistics*, pages 4519–4529. PMLR, 2020.

Jakub Konečný, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *JMLR*, 19(1):3140–3207, 2018.

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

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization for heterogeneous networks. *arXiv preprint*, 2019. URL `https://arxiv.org/pdf/1812.06127.pdf`.

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

Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed Iterate Analysis for Asynchronous Stochastic Optimization. *SIAM Journal on Optimization*, pages 2202–2229, 2015.

Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data, 2017. URL `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. Last accessed 09/24/2019.

H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *ICLR Workshop Track*, 2016.

Qi Meng, Wei Chen, Jingcheng Yu, Taifeng Wang, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic proximal optimization algorithms with variance reduction. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Lam M. Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takác. SGD and hogwild! convergence without the bounded gradients assumption. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 3747–3755, 2018.

Lam M. Nguyen, Phuong Ha Nguyen, Peter Richtárik, Katya Scheinberg, Martin Takáč, and Marten van Dijk. New convergence aspects of stochastic gradient algorithms. *Journal of Machine Learning Research*, 20(176):1–49, 2019a.

P. H. Nguyen, L. M. Nguyen, and M. van Dijk. Tight dimension independent lower bound on the expected convergence rate for diminishing step sizes in SGD. *The 33th Annual Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019b.

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

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Hongchao Zhang Saeed Ghadimi, Guanghui Lan. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *arXiv preprint arxiv:1308.6594*, 2013.

Shaohuai Shi, Qiang Wang, Kaiyong Zhao, Zhenheng Tang, Yuxin Wang, Xiang Huang, and Xiaowen Chu. A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks. *arXiv preprint arXiv:1901.04359*, 2019.

Artin Spiridonoff, Alex Olshevsky, and Ioannis Ch Paschalidis. Local sgd with a communication overhead depending only on the number of workers. *arXiv preprint arXiv:2006.02582*, 2020.

Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

Marten Van Dijk, Lam Nguyen, Phuong Ha Nguyen, and Dzung Phan. Characterization of convex objective functions and optimal expected convergence rates for sgd. In *International Conference on Machine Learning*, pages 6392–6400, 2019.

Luping Wang, Wei Wang, and Bo Li. Cmfl: Mitigating communication overhead for federated learning. *IEEE International Conference on Distributed Computing Systems.*, 2019.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint*, 2019. URL `https://arxiv.org/pdf/1903.03934v1.pdf`.

Hao Yu and Rong Jin. On the computation and communication complexity of parallel sgd with dynamic batch sizes for stochastic non-convex optimization. In *International Conference on Machine Learning*, pages 7174–7183. PMLR, 2019.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4120–4129. JMLR. org, 2017.

Martin Zinkevich, John Langford, and Alex J Smola. Slow learners are fast. In *Advances in neural information processing systems*, pages 2331–2339, 2009.