Sharper Convergence Guarantees for Asynchronous SGD for Distributed and Federated Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study the asynchronous stochastic gradient descent algorithm, for distributed training over n workers that might be heterogeneous. In this algorithm, workers compute stochastic gradients in parallel at their own pace and return them to the server without any synchronization.

Existing convergence rates of this algorithm for non-convex smooth objectives depend on the maximum delay $\tau_{\max}$ and reach an ε-stationary point after $\mathcal{O}\big(\sigma^2\varepsilon^{-2} + \tau_{\max}\varepsilon^{-1}\big)$ iterations, where σ is the variance of stochastic gradients. In this work (i) we obtain a tighter convergence rate of $\mathcal{O}\big(\sigma^2\varepsilon^{-2} + \sqrt{\tau_{\max}\tau_{avg}}\varepsilon^{-1}\big)$ *without any change in the algorithm* where τ_{avg} is the average delay, which can be significantly smaller than $\tau_{\max}$. We also provide (ii) a simple delay-adaptive learning rate scheme, under which asynchronous SGD achieves a convergence rate of $\mathcal{O}\big(\sigma^2\varepsilon^{-2} + \tau_{avg}\varepsilon^{-1}\big)$, and does not require any extra hyperparameter tuning nor extra communications. Our result allows to show *for the first time* that asynchronous SGD is *always faster* than mini-batch SGD. In addition, (iii) we consider the case of heterogeneous functions motivated by federated learning applications and improve the convergence rate by proving a weaker dependence on the maximum delay compared to prior works.

1 Introduction

The stochastic gradient descent (SGD) algorithm [37, 11] and its variants form the foundation of modern machine learning training methods. With recent growth in the sizes of models and available training data, parallel and distributed versions of SGD are becoming increasingly important [50, 15, 14]. Without it, modern state-of-the art language models [38], generative models [34, 35], and many others [44] would not be possible. Distributed SGD is the natural distributed version of classical mini-batch SGD. In the distributed setting, also known as data-parallel training, several workers distribute the optimization over many compute devices (e.g. cores, or GPUs on a cluster) in order to speedup training. Every worker computes gradients on a subset of the training data, and the resulting gradients are aggregated (averaged) on a server.

The same type of SGD variants also form the core algorithms for federated learning applications [30, 22] where the training process is naturally distributed over the user devices, or clients, that keep their local data private, and only transfer the (e.g. encrypted or differentially private) gradients to the server.

A rich literature exists on the convergence theory of above mentioned parallel SGD methods, in particular in the synchronous case, that is when updates from all workers arrive at the same time, see e.g. [15, 11] and references therein. This also includes other orthogonal approaches to make distributed learning and mini-batch SGD more efficient, such as communication compression techniques [2, 3, 42, 43], or performing several local SGD steps on workers before communicating with the server [26, 28, 30, 41].

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

In this paper we consider the more challenging case of *asynchronous* variants of SGD. In classical synchronous mini-batch SGD, workers in each round are required wait for the slowest one, before being able to start the next round of gradients. In asynchronous SGD, each worker starts the next computation immediately after finishing computing its own gradient, without waiting for any other workers. This is especially important in the presence of straggler nodes. Asynchronous algorithms were studied both in distributed and federated learning settings [36, 27, 24, 40, 31].

Most works have studied convergence properties of the asynchronous SGD algorithm for homogeneous distributed settings. This arises for instance in shared-memory implementations where all processes can access the same data [36]. In this setting, it can be proven that asynchronous SGD finds an ε-approximate stationary point (squared gradient norm bounded by ε) in $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{\max}}{\varepsilon}\big)$ iterations [42], for smooth non-convex functions. This complexity bound depends on the maximum delay of the gradients $\tau_{\max}$and the gradient variance $\sigma > 0$. Unfortunately, the maximal delay is a very pessimistic parameter that might not reflect well the true behavior in practice. For instance, if a worker struggles just once, the maximum delay can be large, while we would still expect reasonable convergence.

Two recent works [13, 7] tackle this issue by proposing two new delay-adaptive algorithms that achieve a convergence rate of $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg}}{\varepsilon}\big)$ that depends only on the average delay of the applied gradients.[1] The average delay can be much smaller than the maximal delay, and thus these methods are robust to rare stragglers. However, Cohen et al. [13] requires twice more communications at every step, and an extra hyperparameter to tune. Aviv et al. [7] analyze only convex functions and assume a bound on the variance of the delays, that can frequently degrade with the maximum delay $\tau_{\max}$. Moreover, require to assume that gradients are uniformly bounded.

In the heterogeneous function case, that is in particular relevant in federated learning applications [22], all the existent convergence rates of asynchronous SGD depend on the maximum delay [31].

Contributions.

- For standard asynchronous SGD with constant stepsize, and with non-convex L-smooth homogeneous functions, we prove the tighter convergence rate of $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\sqrt{\tau_{avg}\tau_{\max}}}{\varepsilon}\big)$ to ε-small error. Under the additional assumption of bounded gradients, we obtain $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg}G}{\varepsilon^{3/2}} + \frac{\tau_{avg}}{\varepsilon}\big)$ convergence rate. The previous best known rate was $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{\max}}{\varepsilon}\big)$.
- With homogeneous functions, we provide a delay-adaptive stepsize scheme that does not require tuning of any extra hyperparameters, and converges at the rate of $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg}}{\varepsilon}\big)$ for non-convex L-smooth functions.
- This result allows us to show *for the first time* that asynchronous SGD is always better than mini-batch SGD regardless of the delays pattern (under assumption that the server can perform operations with zero time).
- We also consider distributed optimization with heterogeneous functions where the delays might depend on the nodes and give the convergence rate of $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\zeta^2}{\varepsilon^2} + \frac{\sqrt{\tau_{avg}\frac{1}{n}\sum_{i=1}^n \zeta_i^2\tau_{avg}^i}}{\varepsilon^{\frac{3}{2}}} + \frac{\sqrt{\tau_{avg}\tau_{\max}}}{\varepsilon}\big)$, where ζ_i's measure functions heterogeneity and $\bar{\tau}_i$ is the average delay of the node i. This rate improves the best previously-known results that had worse dependence on the maximum delay $\tau_{\max}$.

2 Related Work

Asynchronous SGD. The research field of asynchronous optimization can be traced back at least to 1989 [9]. Recent works are heavily focused on its SGD variants, such as Hogwild! SGD [33] which deals with coordinate-wise asynchronity. Nguyen et al. [32] provided a tighter convergence analysis by removing the bounded gradient assumption. Our work does not focus on such a coordinate-wise asynchrony as it relies on sparsity assumption that is not realistic in modern machine learning applications. Mania et al. [27] introduces the perturbed iterate framework which enabled theoretical advances with tighter convergence rates [42, 40]. Leblond et al. [24] focus on asynchronous variance-reduced methods.

[1] Aviv et al. [7] considered only the convex cases in their analysis. As we focus in this work on non-convex optimization, we transferred their results in our setting.

Many works [1, 12, 18, 5, 39, 25, 42, 16] focused on asynchronous SGD variants where workers communicate with the server without any synchronization, but these communications are considered to be atomic. All of these works provide convergence guarantees that depend on the maximum delay $\tau_{\max}$ with [5, 42] providing the first tight convergence rates under assumption that the delays are always constant for quadratic and general (convex, strongly convex and non-convex) functions correspondingly. Stich et al. [40] showed a connection of large batches and delays, although still depending only on the maximum delay. Even et al. [17] consider a continuized view of the time (rather than classical per-iteration time) for asynchronous algorithms on a decentralized network.

Delay-adaptive methods. The works [49, 48, 39, 45, 29, 16] considered delay-adaptive schemes to mitigate adversarial effect of stragglers, however with convergence rates that still depend on the maximum delay $\tau_{\max}$. Only Cohen et al. [13] in non-convex and [7] in convex case were able to obtain convergence rates depending on the average delay τ_{avg}.

Asynchronous federated learning. In typical federated learning (FL) applications [30], clients or workers frequently have very different computing powers/speed. This makes especially appealing for practitioners to use asynchronous algorithms for FL [41, 31, 6, 47, 21, 8, 20, 46] with many of these works focusing on correcting for unequal participation ratio of different clients [46, 20, 21, 8, 47] by implementing variance reduction techniques on the server. Nguyen et al. [31] introduce the FedBuff algorithm that is very close to the algorithm that we consider in this work and show its practical superiority over classical synchronous FL algorithms.

3 Setup

We consider optimization problems where the components of the objective function (i.e. the data for machine learning problems) is distributed across n nodes (or clients),

$$\min_{\mathbf{x}\in\mathbb{R}^d}\left[f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n}\left[f_i(\mathbf{x}) = \mathbb{E}_{\xi\sim\mathcal{D}_i}\, F_i(\mathbf{x},\xi)\right]\right]. \tag{1}$$

Here $f_i\colon \mathbb{R}^d \to \mathbb{R}$ denotes the local loss function that is accessible to the node i, $i \in [n] := \{1,\dots n\}$. Each f_i is a stochastic function $f_i(\mathbf{x}) = \mathbb{E}_{\xi\sim\mathcal{D}_i}\, F_i(\mathbf{x},\xi)$ and clients can only access stochastic gradients $\nabla F_i(\mathbf{x},\xi)$. This setting covers deterministic optimization if $F_i(\mathbf{x},\xi) = f_i(\mathbf{x})$, $\forall\xi$. It also covers *empirical risk minimization* problems by setting $\mathcal{D}_i$ being a uniform distribution over a local dataset $\{\xi_i^1 \dots \xi_i^{m_i}\}$ of size m_i. In this case the local functions f_i can be written as finite sums: $f_i(\mathbf{x}) = \frac{1}{m_i}\sum_{j=1}^{m_i} F_i(\mathbf{x},\xi_i^j)$.

Assumptions. For our convergence analysis we rely on following standard assumptions on the functions f_i and F_i:

Assumption 1 (bounded variance)**.** *We assume that there exists a constant $\sigma \geq 0$ such that*

$$\mathbb{E}_{\xi\sim\mathcal{D}_i}\left\|\nabla F_i(\mathbf{x},\xi) - \nabla f_i(\mathbf{x})\right\| \leq \sigma^2\,, \qquad \forall i\in[n], \forall \mathbf{x}\in\mathbb{R}^d\,. \tag{2}$$

Assumption 2 (bounded function heterogeneity)**.** *We assume that there exists n constants $\zeta_i \geq 0$, $i \in [n]$ such that*

$$\left\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\right\|_2^2 \leq \zeta_i^2\,, \quad \forall \mathbf{x}\in\mathbb{R}^d\,, \qquad \textit{and define} \quad \zeta^2 := \tfrac{1}{n}\textstyle\sum_{i=1}^n \zeta_i^2\,. \tag{3}$$

Assumption 3 (L-smoothness)**.** *Each function $f_i\colon \mathbb{R}^d \to \mathbb{R}$, $i \in [n]$ is differentiable and there exists a constant $L \geq 0$ such that*

$$\left\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\right\| \leq L\left\|\mathbf{x}-\mathbf{y}\right\|\,. \qquad \forall \mathbf{x},\mathbf{y}\in\mathbb{R}^d\,. \tag{4}$$

For only *some* of the results we will assume a bound on the gradient norm.

Assumption 4 (bounded gradient)**.** *Each function $f_i\colon \mathbb{R}^d \to \mathbb{R}$, $i \in [n]$ is differentiable and there exists a constant $G \geq 0$ such that*

$$\left\|\nabla f_i(\mathbf{x})\right\|_2^2 \leq G^2\,, \qquad \forall \mathbf{x}\in\mathbb{R}^d\,. \tag{5}$$

Algorithm 1 ASYNCHRONOUS SGD

input Initial value $\mathbf{x}^{(0)} \in \mathbb{R}^d$

1: sever selects a set of active workers $\mathcal{C}_0 \subseteq [n]$ and sends them $\mathbf{x}^{(0)}$
2: **for** $t = 0, \ldots, T-1$ **do**
3: active workers $\mathcal{C}_t$ are computing stochastic gradients in parallel at the assigned points
4: once a worker j_t finishes compute, it sends $\nabla F(\mathbf{x}^{(t-\tau_t)}, \xi_t)$ to the server
5: server updates $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla F(\mathbf{x}^{(t-\tau_t)}, \xi_t)$
6: server selects subset $\mathcal{A}_t \subseteq [n]$ of inactive workers, i.e. $(\mathcal{C}_t \backslash \{j_t\}) \cap \mathcal{A}_t = \emptyset$, and sends them $\mathbf{x}^{(t+1)}$
7: update active worker set $\mathcal{C}_{t+1} = \mathcal{C}_t \backslash \{j_t\} \cup \mathcal{A}_t$
8: **end for**

4 Homogeneous Distributed Setting

We start with an important special case of problem (1) where the components are idential between workers, i.e. $f_i(\mathbf{x}) \equiv f_j(\mathbf{x})$ for all $i, j \in [n]$, such as in the case of homogeneously distributed training data. Consequently, this implies that Assumption 2 holds with $\zeta_i = 0$, $i \in [n]$. Many classical works have focused on asynchronous algorithms under this homogeneous setting (e.g. [5, 42, 1, 18, 39, 25], see the related work for more references). This setting commonly appears in the datacenter setup for distributed training [14], where all nodes (or GPUs) have access to the full dataset or data distributions. Moreover, this special case allows us to present our main ideas in a simplified way, without complicating the presentation due to heterogeneity. We will later see that most of the results in this section can also be obtained as a corollary of the more general heterogeneous functions case (Section 5) by setting $\zeta_i = 0$ $i \in [n]$.

4.1 Algorithm

We consider standard asynchronous SGD (also known as delayed-SGD, or SGD with stale updates) as presented in Algorithm 1, see e.g. [5, 42, 1, 18, 39, 25]. First, the server initializes training by selecting an initial active worker set $\mathcal{C}_0$ and assigning $\mathbf{x}^{(0)}$ to these workers. Throughout the algorithm, the active workers compute gradients at their own speed, based on their local data. On line 4, once some worker (which we denote as j_t) finishes computing its gradient, it sends the result to the server. On line 5 the server incorporates the received—possibly delayed—gradient, using a stepsize η_t that can depend on the gradient delay τ_t. The *gradient delay* τ_t is defined as the difference between the iteration at which worker j_t started to compute the gradient and the iteration t at which it got applied. We index the stochastic noise of the gradients ξ_t by iteration t to highlight that previous iterates $\mathbf{x}^{(t')}$ for $t' \leq t$ do not depend on this stochastic noise. However, the client selects the data sample ξ_t at iteration $t - \tau_t$ when the computation starts. After that, on lines 6-7 the server selects the new active workers out of the ones that are currently inactive (including worker j_t) and assigns them the latest iterate $\mathbf{x}^{(t+1)}$.

In contrast to previous works, we explicitly define the set of workers that are busy with computations at every step t as $\mathcal{C}_t$ (the active workers set). A main advantage of allowing the sets $\mathcal{C}_t$ to be different at every step t lies in the possibility to also cover mini-batch SGD as a special case, which we discuss in Example 2. Our theoretical results depend on the size of these sets $\mathcal{C}_t$, a.k.a. the *concurrency*.

Definition 1 (Concurrency)**.** *The* concurrency $\tau_C^{(t)}$ *at step* t *is defined as the size of the active worker set* $\mathcal{C}_t$*, i.e.* $\tau_C^{(t)} = |\mathcal{C}_t|$*. We also define the maximum and average concurrency as*

$$\tau_C = \max_t \{\tau_C^{(t)}\}\,, \qquad \bar{\tau}_C = \tfrac{1}{T+1} \textstyle\sum_{t=0}^{T} \tau_C^{(t)}\,.$$

Note that in practical scenarios, we have a *constant concurrency* of n over time, meaning that all n workers are active at every step, and thus $\tau_C = \bar{\tau}_C = n$.

We discuss two important practical examples that fit into our Algorithm 1:

Example 2 (Mini-batch SGD)**.** *Mini-batch SGD with batch size* n *can be seen as a special case of Algorithm 1, as follows: The server (i) in line 1 selects all* n *workers,* $\mathcal{C}_0 = [n]$*; (ii) in line 6 does not select new workers while the gradients from the same batch haven't been fully applied yet, i.e.* $\mathcal{A}_t = \emptyset$ *if* $t \bmod n \neq 0$*; (iii) in line 6 selects* $\mathcal{A}_t = [n]$ *if* $t \bmod n = 0$ *to start a new batch.*

Example 3 (Asynchronous SGD with maximum concurrency)**.** *In practical implementations one should always aim to utilize all resources available and thus (i) in line 1 select all available workers $\mathcal{C}_0 = 0$; (ii) in line 6 select the worker that finished its computations $\mathcal{A}_t = \{j_t\}$ so that workers are always busy with jobs.*

4.2 Theoretical analysis: Constant stepsizes

We first formally define the average and maximum delays.

Definition 4 (Average and maximum delays)**.** *Let $\{\tau_t\}_{t=0}^{T-1}$ be the delays of the applied gradients in Algorithm 1. We define $\{\tau_i^{\mathcal{C}_T}\}_{i\in\mathcal{C}_T\setminus\{j_T\}}$ as the delays of gradients which are in flight at time T, that is they have remained unapplied at the last step. Each $\tau_i^{\mathcal{C}_T}$ is equal to the difference between the last iteration T and the iteration at which worker i started to compute its last gradient. We then define the average and the maximum delays as*

$$\tau_{avg} = \frac{1}{T + |\mathcal{C}_T| - 1}\left(\sum_{t=0}^{T-1}\tau_t + \sum_{i\in\mathcal{C}_T\setminus\{j_T\}}\tau_i^{\mathcal{C}_T}\right), \quad \tau_{\max} = \max\left\{\max_{t=1,\dots T-1}\tau_t, \max_{i\in\mathcal{C}_T\setminus\{j_T\}}\tau_i^{\mathcal{C}_T}\right\}. \tag{6}$$

We further provide a key observation on the connection between the average delay and the average concurrency. This observation, is one of the essential elements for achieving an improved analysis.

Remark 5 (Key Observation)**.** *In Algorithm 1 the average concurrency $\bar{\tau}_C$ is connected to the average delay τ_{avg} as*

$$\tau_{avg} = \frac{T+n}{T}\bar{\tau}_C \quad \overset{T\geq n}{=} \quad \mathcal{O}(\bar{\tau}_C)\ . \tag{7}$$

We explain the intuition behind this observation on a simple example. Assume that the concurrency is constant at every step ($\tau_C = \bar{\tau}_C$), and that all workers except one are responding very rarely. Then on steps 4–5 of Algorithm 1 only this one responding worker would mostly participate. This means that for this one worker the delay τ_t would be frequently equal to zero, and the overall average delay will be small.

Next, we provide our theoretical results. We first focus on the Asynchronous SGD Algorithm 1 under constant stepsizes, i.e. $\eta_t \equiv \eta$. This setting was studied in many works such as [1, 18, 5, 25, 42]

Theorem 6 (constant stepsizes)**.** *Under Assumptions 1, 3, there exists a constant stepsize $\eta_t \equiv \eta$ such that for Algorithm 1 it holds that $\frac{1}{T+1}\sum_{t=0}^{T}\left\|\nabla f(\mathbf{x}^{(t)})\right\|_2^2 \leq \varepsilon$ after*

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\sqrt{\tau_C\tau_{\max}}}{\varepsilon}\right) \qquad \textit{iterations}. \tag{8}$$

If we additionally assume bounded gradient Assumption 4, then $\frac{1}{T+1}\sum_{t=0}^{T}\left\|\nabla f(\mathbf{x}^{(t)})\right\|_2^2 \leq \varepsilon$ after

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_C G}{\varepsilon^{3/2}} + \frac{\tau_C}{\varepsilon}\right) \qquad \textit{iterations.} \tag{9}$$

Under constant concurrency, we can directly connect τ_C to the average delay τ_{avg} due to Remark 5. We highlight again that in practice, to get the best utilization of the available resources, practical implementations choose the maximum concurrency possible, which is equal to n.

Corollary 7. *If in Algorithm 1 the concurrency is constant at every step (thus $\tau_C = \bar{\tau}_C$), then under the same conditions as in Theorem 6 the convergence rate of Algorithm 1 is equal to*

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\sqrt{\tau_{avg}\tau_{\max}}}{\varepsilon}\right) \qquad \textit{and} \qquad \mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg} G}{\varepsilon^{3/2}} + \frac{\tau_{avg}}{\varepsilon}\right) \tag{10}$$

for the case without and with bounded gradient Assumption 4 correspondingly.

The previous best-known convergence rate for Asynchronous SGD 1 under constant stepsizes was given in [42] and is equal to $\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{\max}}{\varepsilon}\right)$. In this theorem we improved the delay dependence from $\tau_{\max}$ to $\sqrt{\tau_{avg}\tau_{\max}}$ in the last term *without any change in the algorithm*, only by taking into account concurrency that is usually fixed in practical implementations anyways. No other work previously made an assumption on the number of computing workers. $\sqrt{\tau_{avg}\tau_{\max}}$ could be much smaller than $\tau_{\max}$ in the presence of rare straggler devices. With an additional assumption of bounded gradients, the dependence on the maximum delay can be completely removed.

4.3 Theoretical analysis: Delay-adaptive stepsizes

In many cases, the bounded gradient Assumption 4 is unrealistic [32], meaning that the gradient bound G is often large and thus the rate (9) is loose. In this section we show that by weighting the stepsize down for the gradients that have a large delay, we can remove the dependence on the maximum delay $\tau_{\max}$ without assuming bounded gradients (Assump. 4).

Theorem 8 (delay-adaptive stepsizes)**.** *There exist a parameter $\eta \leq \frac{1}{4L}$ such that if we set the stepsizes in Algorithm 1 dependent on the delays as*

$$\eta_t = \begin{cases} \eta & \tau_t \leq \tau_C, \\ < \min\{\eta, \frac{1}{4L\tau_t}\} & \tau_t > \tau_C, \end{cases} \tag{11}$$

then for Algorithm 1, under Assumptions 1, 3 it holds that $\frac{1}{T+1}\sum_{t=0}^{T}\left\|\nabla f(\mathbf{x}^{(t)})\right\|_2^2 \leq \varepsilon$ after

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_C}{\varepsilon}\right) \quad \textit{iterations.} \tag{12}$$

In our theorem, the stepsize η_t in the case of large delays $\tau_t > \tau_C$ can be an arbitrary value between 0 and $\min\{\eta, \frac{1}{4L\tau_t}\}$. Setting the stepsize $\eta_t \equiv 0$ is equivalent to dropping these gradients.

Proof sketch of Theorem 8. We give the intuitive proof sketch for the case when we drop gradients with $\tau_t > \tau_C$ and we deal with the general case in the Appendix. We know that $\tau_{avg} \approx \bar{\tau}_C \leq \tau_C$ from Remark 5. It also holds that the number of gradients that have delay larger than the average delay τ_{avg} is smaller than half of all the gradients ($\leq \frac{T}{2}$) because delays are bounded below by zero ($\tau_t \geq 0\ \forall t$). Thus, dropping the gradients with the delay $\tau_t > \tau_C$, or equivalently setting their stepsize $\eta_t \equiv 0$, will degrade the convergence rate at most by half, while the maximum delay among the applied ones now is equal to τ_C. Thus we can apply result from [42] with $\tau_{max} = \tau_C$. □

Corollary 9. *If in Algorithm 1 the concurrency is constant at every step (thus $\tau_C = \bar{\tau}_C$), then under the same conditions as in Theorem 8 the convergence rate of Algorithm 1 is equal to*

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg}}{\varepsilon}\right). \tag{13}$$

4.4 Discussion

Comparison to synchronous optimization. Mini-batch SGD with batch size n has the same degree of parallelism as Algorithm 1 with constant concurrency n, i.e. it has n workers computing gradients in parallel. Mini-batch SGD needs $\mathcal{O}\left(\frac{\sigma^2}{n\varepsilon^2} + \frac{1}{\varepsilon}\right)$ [19] batches of gradients to reach an ε-stationary point, and thus needs $\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{n}{\varepsilon}\right)$ gradients, as the batch-size is equal to n. On the contrary, asynchronous SGD Algorithm 1 with stepsizes chosen as in (11) achieves exactly the same rate (13) since $\tau_{avg} = \tau_C = n$, while its expected per-iteration time is *faster* than that of mini-batch SGD, as no workers have to wait for others. Thus, our result shows *for the first time* that asynchronous SGD is always faster than mini-batch SGD, regardless of the delay pattern. A small note that in our reasoning we implicitly assumed that the sever can perform its operations in negligible time.

Tuning the stepsize. It is worth noting that our stepsize rule (11) does not introduce any additional hyperparameters to tune compared to the constant stepsize case or to synchronous SGD. τ_C is usually known and can be easily controlled by the server, especially in the practical constant concurrency case. Thus, to implement such a stepsize rule (11) one needs to tune only stepsize η, and in case of $\tau_t > \tau_C$ set stepsize $\eta_t \leq \frac{\eta}{\tau_t}$.

Average vs maximum delay. In a homogeneous environment when every worker computes gradients with same speed during the whole training, the average and maximum delays would be almost equal. However, occasional straggler devices will usually be present. In this case the maximum delay is much larger than the average delay.

Consider a simple example with $n = 2$ workers, where the first worker computes gradients very fast, while the second worker returns its gradient only at the end of the training at the last iteration T. In this case the average delay $\tau_{avg} = 2$ is a small constant, while the maximum delay $\tau_{\max} = T$. In this case the rate depending only on the maximum delay $\tau_{\max}$ would guarantee convergence only up to a constant accuracy $\varepsilon = \mathcal{O}(1)$. While both rates with $\sqrt{\tau_{\max}\tau_{avg}}$ and with τ_{avg} guarantee convergence up to an arbitrary small accuracy.

Comparison to other methods. [13] recently proposed the PickySGD algorithm that achieves (same as (13)) a convergence rate of $\mathcal{O}\big(\frac{\sigma^2}{\varepsilon^2} + \frac{\tau_{avg}}{\varepsilon}\big)$. Their algorithm discards gradients based on the distance between the current point and the delayed one $\big\|\mathbf{x}^{(t)} - \mathbf{x}^{(t-\tau_t)}\big\|$. The disadvantage of their method is that it requires sending points $\mathbf{x}^{(t-\tau_t)}$ along with the gradients thus incurring twice more communications at every step. Their method also requires tuning an extra hyperparameter. In this work we achieve the same convergence rate with a much simpler method that does not require any additional communications nor additional tuning compared to synchronous SGD.

[7] also recently proposed the delay-adaptive algorithm with convergence rate depending on the average delay τ_{avg} for the convex and strongly convex cases. Although, our convergence rates are for the non-convex case and are not directly comparable to theirs, we highlight some key differences in their analysis. First, their convergence rate depends not only on τ_{avg} but also on the variance σ_τ of the delays, which can degrade with the maximum delay. Second, they require the bounded gradient Assumption 4. In Theorem 6 we show that under Assumption 4 no modifications to the algorithm are needed to completely remove the dependence on the maximum delay $\tau_{\max}$ (9).

Tightness. As we explained in Example 2, mini-batch SGD is covered by Algorithm 1. We know that mini-batch SGD convergence is lower bounded by $\Theta\big(\frac{\sigma^2}{n\varepsilon^2} + \frac{1}{\varepsilon}\big)$ [4] in terms of batches processed and thus by $\Theta\big(\frac{\sigma^2}{\varepsilon^2} + \frac{n}{\varepsilon}\big)$ in terms of the gradients computed. Our convergence rate given in Theorem 6 *coincides with this lower bound* as in this case concurrency $\tau_C = n$, $\tau_{avg} = \bar{\tau}_C = \frac{n}{2}$.

5 Heterogeneous Distributed Setting

In this section we consider more general problems of the form (1) where the functions f_i are different on different nodes. This setting is motivated for example by federated learning [30, 22], where every node (client) possesses its own private data, possibly coming from a different data distribution, and thus has its own different local loss function f_i.

The setting here is therefore more general than the one considered in previous Section 4, and we will see that some of the results (with the constant stepsizes) in the homogeneous case follow as a special case of the more general results we present in this section.

5.1 Algorithm

We consider asynchronous SGD algorithm given in Algorithm 2. Close variants of this algorithm were studied in several prior works [31, 41]. In order to simplify the presentation, we consider that concurrency is constant over time (and thus $\tau_C = \bar{\tau}_C$ in Definition 1). In order to allow for client subsampling often implemented in practical FL applications, we allow the concurrency τ_C to be smaller than overall number of workers n. The same concurrency model was recently considered in the practical FedBuff algorithm [31].

Algorithm 2 ASYNCHRONOUS SGD with concurrency τ_C

input Initial value $\mathbf{x}^{(0)} \in \mathbb{R}^d$, n clients, concurrency τ_C

Server:

1: sever selects *uniformly at random* a set of active clients $\mathcal{C}_0$ of size τ_C and sends them $\mathbf{x}^{(0)}$
2: **for** $t = 0, \ldots, T-1$ **do**
3: active clients $\mathcal{C}_t$ are computing stochastic gradients in parallel at the assigned points
4: once some client j_t finishes compute, it sends $\nabla F_{j_t}(\mathbf{x}^{(t-\tau_t)}, \xi_t)$ to the server
5: server updates $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla F_{j_t}(\mathbf{x}^{(t-\tau_t)}, \xi_t)$
6: sever selects a new client $k_t \sim \mathrm{Uniform}[1, n]$ and sends it $\mathbf{x}^{(t+1)}$
7: update the active worker multiset $\mathcal{C}_{t+1} = \mathcal{C}_t \backslash \{j_t\} \cup \{k_t\}$
8: **end for**

The algorithm is very similar to the homogeneous Algorithm 1 with two key differences: at line 6, the server selects clients *out of all clients*, and does so *uniformly at random*, regardless of the current active worker set $\mathcal{C}_t$. This means that the same client can get sampled several times, even if it didn't finish its previous job(s) yet (thus $\mathcal{C}_t$ is a multiset). In this case, the assigned jobs would just pile up on this client.

5.2 Theoretical analysis

We first note that our key observation on the delays (Remark 5) holds for Algorithm 2 as well. Moreover, as we have a constant concurrency τ_C at every step, $\tau_{avg} = \mathcal{O}(\tau_C)$.

Theorem 10 (constant stepsizes)**.** *Under Assumptions 1, 2, 3, there is exist a constant stepsize $\eta_t \equiv \eta$ such that for Algorithm 2 it holds that $\frac{1}{T+1}\sum_{t=0}^{T}\left\|\nabla f(\mathbf{x}^{(t)})\right\|_2^2 \leq \varepsilon$ after*

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\zeta^2}{\varepsilon^2} + \frac{\sqrt{\tau_{avg}\frac{1}{n}\sum_{i=1}^{n}\zeta_i^2\tau_{avg}^i}}{\varepsilon^{\frac{3}{2}}} + \frac{\sqrt{\tau_{avg}\tau_{\max}}}{\varepsilon}\right) \qquad \textit{iterations,} \tag{14}$$

where τ_{avg}^i is the average delay of the gradients from client i. If we additionally assume bounded gradient Assumption 4, then $\frac{1}{T+1}\sum_{t=0}^{T}\left\|\nabla f(\mathbf{x}^{(t)})\right\|_2^2 \leq \varepsilon$ after

$$\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\zeta^2}{\varepsilon^2} + \frac{\tau_{avg}G}{\varepsilon^{\frac{3}{2}}} + \frac{\tau_{avg}}{\varepsilon}\right) \qquad \textit{iterations.} \tag{15}$$

We note that the leading $\frac{1}{\varepsilon^2}$ term is affected by heterogeneity ζ^2 because at every step we apply gradient from only one client. This term is usually present in the federated learning algorithms with client subsampling see e.g. [23].

5.3 Discussion

Comparison to other works. The recent FedBuff algorithm [31] is similar to our Algorithm 2. Their algorithm allows clients to perform several local steps and the server to wait for more than 1 client to finish compute (aka buffering), which we did not include for simplicity as these aspects are orthogonal to the effect of delays.

Disregarding these two orthogonal changes, the FedBuff algorithm is almost equivalent to our Algorithm 2 with a key difference: they assume that the client j_t that finishes computation at every step comes from the uniform distribution over all the clients. This is unrealistic to assume in practice because the server cannot control which clients finish computations at every step. In Algorithm 2 we have the more realistic assumption that only on the sampling process of the clients (on line 6) *can be controlled* by the server. This reflects practical client sampling in federated learning.

The convergence rate of FedBuff [31] under the bounded gradient assumption is $\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\zeta^2}{\varepsilon^2} + \frac{(\zeta^2+1)\tau_{\max}G^2}{\varepsilon}\right)$. In contrast, in Theorem 10 we completely remove the dependence on the maximum delay $\tau_{\max}$ under bounded gradients (as in Equation (15)).

Delays. We note that for Theorem 10 we did not impose any assumption on the delays. Thus, our result allows clients and the delays on these clients to be dependent, meaning that some of the clients could be systematically slower than others. Interestingly, the middle heterogeneity term (the term with ζ_i) is not affected by the maximum delay at all, but is affected by the average delay within each individual client. If all the heterogeneity parameters are equal, i.e. $\zeta_i = \zeta_j, \forall i, j$, then the middle term will be affected only by the overall average delay τ_{avg}.

Gradient clipping. Practical implementations of FL algorithms usually apply clipping to the gradients in order to guarantee differential privacy [22]. This automatically bounds the norm of all applied gradients, making the the constant G^2 in Assumption 4 small. Although we do not provide formal convergence guarantees of asynchronous SGD with gradient clipping, we envision that its convergence rate would depend only on the average delay, similar to the bounded gradient case (9), thus making the algorithm robust to stragglers.

Delay-adaptive stepsizes. For homogeneous functions we have shown that delay-adaptive stepsizes result in a convergence rate dependent only on the average delay τ_{avg} without assuming bounded gradients (as in Equation (11)). However in the heterogeneous case this is not so straightforward. Delay-adaptive learning rate schemes will introduce a bias towards the clients that compute quickly, and Algorithm 2 would converge to the wrong objective.

It is interesting to note that current popular schemes implemented in practice for FL over-selects the clients at every iteration [10]. The server waits only for some percentage (e.g. 80%) of sampled clients and discards the rest. Such a scheme also introduces a bias towards fast workers. A delay-adaptive

learning rate scheme is expected to introduce less bias as the gradients are still applied but with the smaller weight. We leave this question for future practical investigations, as it is not the focus of our current work.

Independent delays. If the delays and the clients are independent (e.g. coming from the same distribution for all of the clients), then the convergence rate of Algorithm 2 will simplify to $\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{\zeta\tau_{avg}}{\varepsilon^{\frac{3}{2}}} + \frac{\sqrt{\tau_{avg}\tau_{\max}}}{\varepsilon}\right)$ (without needing bounded gradient assumption). In this case it is also possible to use delay-adaptive stepsizes (similar to Theorem 8) to completely remove the dependence on the maximum delays $\tau_{\max}$ without assuming bounded gradients.

Extensions. We can extend the Algorithm 2 and our theoretical analysis to allow clients to perform several local steps, before sending back the change in $\mathbf{x}$. We can also extend Algorithm 2 to allow the server to wait for the first K clients to finish computations rather than just one, similar to [31]. These extensions are straightforward and we excluded them here for simplicity of presentation.

Finally, we can also extend Algorithm 2 to sample new clients as soon as some previous client finished compute, without waiting for the server update on the line 5.

5.4 Estimating Speedup Over the Synchronous SGD

Assume we have n clients, each of which having a different but constant time to compute a gradient $\{\Delta_i\}_{i=1}^n$. W.l.o.g. we assume that Δ_i are ordered as $\Delta_1 \leq \Delta_2 \leq \cdots \leq \Delta_n$.

Lemma 11. *In expectation, the asynchronous Algorithm 2 needs*

$$\bar{\Delta} = \frac{1}{n}\sum_{i=1}^{n}\Delta_i$$

time to compute C gradients, while mini-batch SGD with batch size C needs

$$\tilde{\Delta} = \sum_{i=1}^{n}\alpha_i\Delta_i$$

time to compute a batch of C gradients, where $\alpha_i = \frac{i^C-(i-1)^C}{n^C}$. It is also always holds that $\bar{\Delta} \leq \tilde{\Delta}$.

With this lemma we can precisely estimate how much faster the asynchronous algorithm is compared to the classic synchronous mini-batch one. Note that α_i are increasing with i with a rate of $\mathcal{O}(i^C)$, thus in mini-batch SGD, the large delays get a much higher weight than the small delays, especially when the batch size C is large.

For example, consider 1000 clients, 900 of which compute their update every 10s, while 100 of them computes their update every $60s$. Then the expected time for C gradients of the asynchronous algorithm will be 15s, while synchronous mini-batch SGD (with $C = 10$) will take a significantly longer time of 42.5s for the same number of gradients.

6 Conclusion

In this paper we study the asynchronous SGD algorithm both in homogeneous and heterogeneous settings. By leveraging the concurrency—number of workers that compute gradients in parallel—we show a much faster convergence rate of asynchronous SGD improving the dependence on the maximum delay $\tau_{\max}$ over the prior works, for both homogeneous and heterogeneous cases. Our proof technique also allows to design a simple delay-adaptive stepsize rule (11) that attains a convergence rate depending only on the average delay τ_{avg} that neither require any additional tuning, nor additional communication. Our techniques allows us to demonstrate for the first time that *asynchronous SGD is faster than mini-batch SGD for any delay pattern*.

References

[1] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems 24*, pages 873–881. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4247-distributed-delayed-stochastic-optimization.pdf.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NIPS - Advances in Neural Information Processing Systems 30*, pages 1709–1720. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6768-qsgd-communication-efficient-sgd-via-gradient-quantization-and-encoding.pdf.

[3] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cedric Renggli. The convergence of sparsified gradient methods. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *NeurIPS - Advances in Neural Information Processing Systems 31*, pages 5977–5987. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7837-the-convergence-of-sparsified-gradient-methods.pdf.

[4] Yossi Arjevani, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake E. Woodworth. Lower bounds for non-convex stochastic optimization. *ArXiv*, abs/1912.02365, 2019.

[5] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In Aryeh Kontorovich and Gergely Neu, editors, *Proceedings of the 31st International Conference on Algorithmic Learning Theory*, volume 117 of *Proceedings of Machine Learning Research*, pages 111–132. PMLR, 08 Feb–11 Feb 2020. URL https://proceedings.mlr.press/v117/arjevani20a.html.

[6] Dmitrii Avdiukhin and Shiva Kasiviswanathan. Federated learning under arbitrary communication patterns. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 425–435. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/avdiukhin21a.html.

[7] Rotem Zamir Aviv, Ido Hakimi, Assaf Schuster, and Kfir Yehuda Levy. Learning under delayed feedback: Implicitly adapting to gradient delays. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021.

[8] Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementation of an asynchronous optimization algorithm for the parameter server, 2016. URL https://arxiv.org/abs/1610.05507.

[9] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019. URL https://proceedings.mlsys.org/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf.

[11] L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. URL https://doi.org/10.1137/16M1080173.

[12] Sorathan Chaturapruek, John C Duchi, and Christopher Ré. Asynchronous stochastic convex optimization: the noise is in the noise and SGD don't care. In *Advances in Neural Information Processing Systems 28*, pages 1531–1539. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/6031-asynchronous-stochastic-convex-optimization-the-noise-is-in-the-noise-and-sgd-dont-care.pdf.

[13] Alon Cohen, Amit Daniely, Yoel Drori, Tomer Koren, and Mariano Schain. Asynchronous stochastic optimization robust to arbitrary delays. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9024–9035. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/4b85256c4881edb6c0776df5d81f6236-Paper.pdf.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS - Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[15] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Resesearch (JMLR)*, 13(1):165–202, 2012. URL http://dl.acm.org/citation.cfm?id=2503308.2188391.

[16] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 803–812. PMLR, 2018. URL http://proceedings.mlr.press/v84/dutta18a/dutta18a.pdf.

[17] Mathieu Even, Hadrien Hendrikx, and Laurent Massoulie. Decentralized optimization with heterogeneous delays: a continuous-time approach, 2021. URL https://arxiv.org/abs/2106.03585.

[18] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61(12):3740–3754, Dec 2016. ISSN 0018-9286.

[19] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM J. Optim.*, 23:2341–2368, 2013.

[20] Margalit Glasgow and Mary Wootters. Asynchronous distributed optimization with stochastic delays, 2020. URL https://arxiv.org/abs/2009.10717.

[21] Xinran Gu, Kaixuan Huang, Jingzhao Zhang, and Longbo Huang. Fast federated learning in the presence of arbitrary device unavailability. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12052–12064. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/64be20f6dd1dd46adf110cf871e3ed35-Paper.pdf.

[22] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[23] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019. URL http://arxiv.org/abs/1910.06378.

[24] Remi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *Journal of Machine Learning Research*, 19(81):1–68, 2018. URL http://jmlr.org/papers/v19/17-650.html.

[25] 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.

[26] Olvi L. Mangasarian and Mikhail V. Solodov. Backpropagation convergence via deterministic nonmonotone perturbed minimization. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1994.

[27] 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*, 27(4):2202–2229, 2017. doi: 10.1137/16M1057000. URL https://doi.org/10.1137/16M1057000.

[28] Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464. Association for Computational Linguistics, 2010.

[29] Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5cce8dede893813f879b873962fb669f-Paper.pdf.

[30] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016. URL http://arxiv.org/abs/1602.05629.

[31] John Nguyen, Kshitiz Malik, Hongyua Zhan, Ashka Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, 2022.

[32] Lam Nguyen, PHUONG HA NGUYEN, Marten van Dijk, Peter Richtarik, Katya Scheinberg, and Martin Takac. SGD and hogwild! Convergence without the bounded gradients assumption. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3750–3758. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/nguyen18c.html.

[33] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 693–701. Curran Associates Inc., 2011. URL http://dl.acm.org/citation.cfm?id=2986459.2986537.

[34] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[35] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[36] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *NIPS - Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf.

[37] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951.

[38] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[39] Suvrit Sra, Adams Wei Yu, Mu Li, and Alex Smola. Adadelay: Delay adaptive distributed stochastic optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 957–965. PMLR, 2016. URL http://proceedings.mlr.press/v51/sra16.html.

[40] Sebastian Stich, Amirkeivan Mohtashami, and Martin Jaggi. Critical parameters for scalable distributed learning with large batches and asynchronous updates. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 4042–4050. PMLR, 2021. URL http://proceedings.mlr.press/v130/stich21a/stich21a.pdf.

[41] Sebastian U. Stich. Local SGD converges fast and communicates little. *ICLR - International Conference on Learning Representations*, art. arXiv:1805.09767, 2019. URL https://arxiv.org/abs/1805.09767.

[42] Sebastian U. Stich and Sai Praneeth Karimireddy. The error-feedback framework: Sgd with delayed gradients. *Journal of Machine Learning Research*, 21(237):1–36, 2020. URL http://jmlr.org/papers/v21/19-748.html.

[43] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 1626–1636. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/7434-accelerated-stochastic-matrix-inversion-general-theory-and-speeding-up-bfgs-rules-for-faster-second-order-optimization.pdf.

[44] Meng Wang, Weijie Fu, Xiangnan He, Shijie Hao, and Xindong Wu. A survey on large-scale machine learning. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[45] Xuyang Wu, Sindri Magnusson, Hamid Reza Feyzmahdavian, and Mikael Johansson. Delay-adaptive step-sizes for asynchronous learning, 2022. URL https://arxiv.org/abs/2202.08550.

[46] Yikai Yan, Chaoyue Niu, Yucheng Ding, Zhenzhe Zheng, Fan Wu, Guihai Chen, Shaojie Tang, and Zhihua Wu. Distributed non-convex optimization with sublinear speedup under intermittent client availability, 2020. URL https://arxiv.org/abs/2002.07399.

[47] Haibo Yang, Xin Zhang, Prashant Khanduri, and Jia Liu. Anarchic federated learning, 2021. URL https://arxiv.org/abs/2108.09875.

[48] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016. URL https://www.ijcai.org/Proceedings/16/Papers/335.pdf.

[49] 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*. PMLR, 2017.

[50] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *NIPS - Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We do not propose a new algorithms.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 3
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]