HGO: Robust Distributed Learning with Heterogeneous Participants

Anonymous authors
Anonymous affiliation

December 13, 2021

Abstract

The increasing prevalence of personal devices motivates the design of algorithms that can leverage their computing power, together with the data they generate, in order to build privacy-preserving and effective machine learning models. Traditional distributed learning algorithms however impose a uniform workload on all participating devices, most often discarding the weakest participants. This not only induces a suboptimal use of available computational resources, but also significantly reduces the quality of the learning process, as data held by the slowest devices is discarded from the procedure.

This paper proposes HGO, a distributed learning scheme with parameterizable iteration costs that can be adjusted to the computational capabilities of different devices. HGO encourages the participation of slower devices, thereby improving the accuracy of the model when the participants do not share the same dataset. HGO tolerates Byzantine behavior, by combining a randomized sampling of devices with a robust aggregation method. We demonstrate the convergence of HGO, theoretically and empirically, without assuming any specific partitioning of the data over the devices. We present an exhaustive set of experiments evaluating the performance of HGO on several classification tasks, highlighting the importance of incorporating slow devices when learning, in a Byzantine-prone environment with heterogeneous participants. We also propose an Android implementation to deploy HGO on actual smartphones.

1 Introduction

Mobile devices are now an essential component of our modern society. The number of their users will reach 5.1 billion by 2025, representing 70% of the world's population.[1] Each day, these devices generate a massive amount of data. Rather than letting intermediary companies exploit the data, it is appealing to have each device locally process the data it has generated, both for infrastructure cost efficiency and privacy reasons. State-of-the-art distributed machine learning schemes do enable devices (usually called workers) to collaboratively learn a common model by exchanging information about the data, possibly with the help of a machine server [1, 21]. However, these schemes typically assume participating devices with similar hardware specifications, thereby imposing the same workload on every worker. While this requirement might be satisfied in data centers equipped with powerful co-located computers, steadily connected to the power grid as well as to the network, personal computing devices owned by general public, such as smartphones or laptops, often behave in an unpredictable manner. They may often disconnect from the network, become frequently inactive to save battery consumption, or crash accidentally. Traditional distributed machine learning algorithms are ill-suited for such handheld devices: synchronous algorithms usually show a stable convergence, but the waiting time is dictated by the weakest device. Asynchronous variants are indeed faster, but they induce high variance in the updates, leading to poor solutions [26]. One can accelerate the training simply by discarding

[1] According to the Mobile Economy 2019 report by the GSM Association:
`https://data.gsmaintelligence.com/api-web/v2/research-file-download?id=39256194&file=2712-250219-ME-Global.pdf`

weaker devices (with low computational power) from the learning process. Whilst this usually has little impact if all workers share the same dataset (a.k.a. the homogeneous case), it can significantly impact the quality of the learning with heterogeneous data. Besides, some devices, technically referred to as *Byzantine* [22], might get hacked and manipulated by malicious parties with the goal of disrupting the system.

In short, three critical issues need to be considered when deploying distributed ML algorithms on handheld devices: the *computational heterogeneity* of the devices, the *heterogeneity of their datasets*, and the possible *Byzantine behavior* of a fraction of them. In this paper, we present HGO, a new distributed learning protocol that addresses all of these three issues. More specifically our technical contribution is twofold.

1. We present a non-trivial optimization protocol, called HGO, that adapts the computational workload to the capabilities of the devices, tackling (directly) computational heterogeneity and (indirectly) data heterogeneity, resulting in a more efficient use of available computing resources and improved learning accuracy. To tolerate Byzantine devices, HGO fuses a robust *aggregation rule* [5, 7, 39, 40] with a random sampling of the workers.

2. We provide theoretical and empirical evidence that HGO can learn accurate models even in a demanding environment that combines heterogeneous and Byzantine participants. In particular, our results highlight the importance of incorporating slow devices when learning in this challenging environment; essentially demonstrating the superiority of HGO over traditional schemes such as SGD in this context.

For presentation simplicity, and to focus on HGO's originality, we do not discuss in this paper how to tolerate Byzantine failures of the parameter server, which we assume to be unique and trusted [1, 21, 27] (a very common assumption in Byzantine machine learning [2, 5, 7, 8, 9, 10, 42, 43]). However, note that HGO can be used with the replication scheme of [13] to circumvent that assumption.

Overview of the protocol. Essentially, HGO can be viewed as a robust, heterogeneity-aware variant of a stochastic *block coordinate descent* algorithm. The server initiates each iteration by randomly choosing a subset of q workers, and sending them its current estimates of the learning parameters. Every honest (non-Byzantine) worker responds with a fragment (i.e., some coordinates) of its mini-batch stochastic gradient. Upon receiving these information, the server identifies a block of coordinates that have been computed by a sufficiently large quorum of at least v workers, and applies a robust aggregation rule on this block to update its current parameter estimates. The Byzantine tolerance of HGO depends upon the underlying aggregation rule. We use *Trimmed Mean* [43] as a default aggregation rule, tolerating up to $f < \frac{v}{2}$ Byzantine workers. Interestingly, any coordinate-wise robust aggregation rule can be used instead.

Analysis and benefits. We prove linear and sub-linear convergence of HGO in the strongly convex and non-convex settings, respectively. We also report empirically on its learning performance on several classification tasks. In case our solution is deployed using actual smartphones, we developed an application running on Android operating systems. We include a quick tutorial on how to setup the application in the extended version of the paper, as well as in the GitHub repository. Our key theoretical and empirical findings demonstrate that, maybe surprisingly, it is possible to incorporate slow devices in the learning procedure without slowing down the protocol by de-correlating the workload of the workers from the computational power of the strongest devices. We also demonstrate that improving the slow device participation considerably outperforms traditional schemes in terms of final accuracy of the model, when the dataset is not shared by all the workers, as illustrated in Figure 1 below. This is achieved without hampering the Byzantine resilience of the process.

Importantly, our analysis highlights an interplay between the convergence of HGO (convergence rate, statistical error), its Byzantine resilience, and the heterogeneity level of the devices. In particular, we demonstrate a critical tension between the computational capabilities of the workers and the impact Byzantine workers can have on the final accuracy of the model. Basically, the more weak devices we have, the more Byzantine workers can affect the final accuracy.

Our protocol and its analysis can be applied to a large family of gradient-based optimization schemes (including CD [38], SCD [36], Block CD [33], GD [30], SGD [15]) as well as to a large set of robust aggregation rules (including Trimmed Mean [43], Aksel [7], MeaMed[39]). To the best of our knowledge, none of these learning algorithms has been analyzed under the combination of the three major issues we consider:

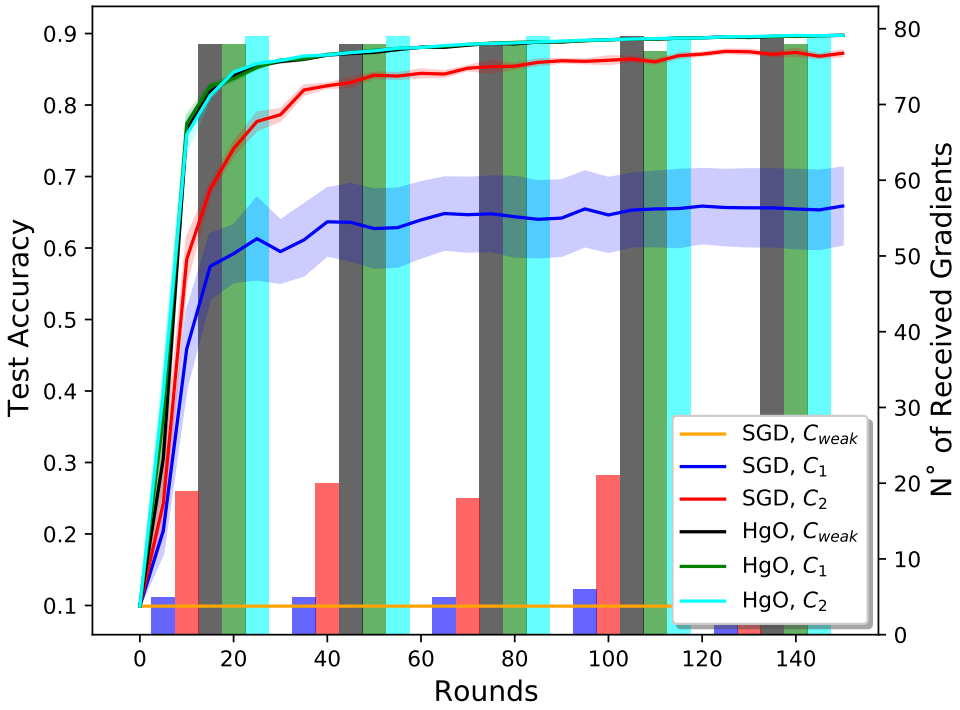


Figure 1: Training a multinomial logistic regression with HgO on MNIST using 100 workers including multiple hardware profiles (C_{weak}, C_1 and C_2). All devices have low computational capabilities in C_{weak}, 90% low and 10% average in C_1, and 70% low and 30% average in C_2. HgO averages the estimates from 80 workers sampled at random at each iteration. The server receives gradients (partial or full) from all the workers regardless of their computational profile. In contrast, with SGD, the server only receives full gradients from strong workers. Clearly, HgO outperforms SGD on both final accuracy and convergence rate.

computational heterogeneity, *data heterogeneity*, and *Byzantine behavior*. Furthermore, HgO is fairly general and could easily be extended with more advanced solvers, such as NAG [29], RMSprop [16], AdaGrad [12], Adam [19], etc.

Roadmap. The rest of the paper is organized as follows. We present the model and preliminary concepts in Section 2. Section 3 describes our HgO algorithm and its time complexity. Section 4 presents our theoretical analysis of HgO and highlights some interesting trade-offs. We present in Section 5 a selection of experiments demonstrating the practical relevance of HgO. Finally, Section 6 discusses related works, and Section 7 concludes the paper. For space limitations, we defer all the proofs, as well as an exhaustive set of experiments, to the full version of the paper, which is available at: `https://anonymous.4open.science/r/HgO-5136`. We also provide an open-source Android implementation of HgO, available at: `https://anonymous.4open.science/r/HgOApp-CC8E`.

2 Preliminaries

We consider the parameter-server architecture [27], where a server orchestrates the training of a machine learning model with the help of a set of n workers. In this setup, the dataset is composed of N data points $X = \{x_1, \ldots, x_N\}$ arbitrarily partitioned over the n workers. The set of data points held by a worker i is denoted by $\mathcal{S}_i$. Thus, $\bigcup_{i=1}^{n} \mathcal{S}_i = X$, and $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for all $i \neq j$. The server is assumed to follow the algorithm correctly, but some of the workers may be faulty, and their identity is a priori unknown [22]. Specifically, there are two types of workers:

1. *Correct workers:* These participants, a.k.a. *honest workers*, correctly follow the instructions prescribed by the server. We represent the correct workers by the set $\mathcal{C} \subseteq [n]$, where $[n]$ denotes the set $\{1, \ldots, n\}$.

2. *Byzantine workers:* Workers that are not honest, i.e., $[n] \setminus \mathcal{C}$, are assumed to be Byzantine. This means that they have an arbitrary behavior and can either crash, or inject adversarial perturbations in the system (see, e.g., [5]).

Our goal is to design a distributed learning algorithm that allows all correct workers to learn a model over the union of their data points $X_{\mathcal{C}} = \bigcup_{i \in \mathcal{C}} \mathcal{S}_i$, despite the presence of up to f Byzantine workers. Specifically, we fix a learning model parameterized by $w \in \mathbb{R}^d$, for which each data point x incurs a loss of $\ell(w, x)$, where the function $\ell : \mathbb{R}^d \times X \to \mathbb{R}$ is assumed differentiable. The algorithm aims to find a local minimum of the function

$$\mathcal{L}(w) := \frac{1}{|X_{\mathcal{C}}|} \sum_{x \in X_{\mathcal{C}}} \ell(w, x). \tag{1}$$

Besides the presence of faulty workers, another challenge we consider in our model is the heterogeneous computational capabilities of the workers. To formally discuss this issue, we first briefly describe the standard distributed stochastic gradient descent (SGD) scheme below.

2.1 Distributed SGD

Distributed SGD is an iterative algorithm where the server maintains a parameter estimate, which is updated iteratively with the help of the workers. We assume that time proceeds in discrete steps, and that each part of the algorithm's execution is allocated a given amount of time units. The initial estimate w_1 may be chosen arbitrarily. In each iteration $k \geq 1$, the server broadcasts its current estimate w_k to all workers, and waits during a fixed amount of time units for the workers to return their stochastic gradients computed at w_k. Each worker i samples a random *mini-batch* $\zeta_{i_k} \subset \mathcal{S}_i$ of size s_i to compute the *stochastic* gradient:

$$G_i(w_k) = \frac{1}{s_i} \sum_{x \in \zeta_{i_k}} \nabla \ell(w_k, x) \tag{2}$$

Then, the workers send back their stochastic gradients to the server, which updates the model parameters as follows:

$$w_{k+1} = w_k - \gamma \left(\frac{1}{n} \sum_{i=1}^{n} G_i(w_k) \right) \tag{3}$$

where $\gamma > 0$ is the learning rate. The algorithm can be proven to eventually output a solution to the learning problem [15]. However, the convergence of Distributed SGD often relies on the assumption that all workers have comparable (homogeneous) computational capabilities and hardware specifications, which is unrealistic when deploying distributed learning on low-powered hand-held devices, such as tablets or smartphones.

2.2 Device heterogeneity: computational power and data

We consider a setting where all devices might not complete the computation of their gradient estimate in the same numbers of time units. Given a parameter w and a data point $x \in \mathcal{S}_i$, we assume that every worker will compute its gradient coordinate by coordinate, at its own pace. Specifically, every worker i can compute $\lambda_i > 0$ coordinates of gradient $\nabla \ell(w, x)$ per time units. Then, in a given time frame of τ time units, worker i can compute $\min\{\frac{\lambda_i \tau}{s_i}, d\}$ coordinates of its stochastic gradient. We call λ_i the computational capability of worker i.

Then, for a fixed mini-batch size, slower workers (with lower computational capabilities) may only be able to compute their stochastic gradients partially, while the faster workers (with higher computational

capabilities) may compute all the coordinates of their stochastic gradients. In what follows, we denote by $\mathcal{Q}_j$ the set of workers that computed coordinate j during a given time frame.

Besides the differences in workers' computational capabilities, our setting faces *data heterogeneity*. Indeed, the data partitioning among the workers may not be uniform. Thus, even the correct workers are unable to compute unbiased estimates of the true gradient $\nabla \mathcal{L}(w)$. This motivates the introduction of Assumptions 3 and 4 in Section 2.4.

2.3 Robust aggregation

In the presence of Byzantine workers, the server cannot rely on the simple averaging of all workers' gradients, as in (3). It should rather use a coordinate-wise robust *aggregation rule* AR, which we formally define below. In the following, for any vector $u \in \mathbb{R}^d$, its j-th coordinate is denoted by $\{u\}_j$.

Definition 1. *For a given $\rho \in [n]$, $w \in \mathbb{R}^d$, a real-valued aggregation rule* AR *is said to be $\Delta(\rho, f)$-*robust *if, for every subset $\mathcal{Q} \subseteq [n]$ with $|\,\mathcal{Q}\,| \geq \rho$ and for every $j \in [d]$, we have*

$$\mathbb{E}\left[\left|\{\mathrm{AG}_w\}_j - \{\nabla \mathcal{L}(w)\}_j\right|^2\right] \leq \Delta(\rho, f)\left(\frac{1}{|\mathcal{C} \cap \mathcal{Q}|}\sum_{i \in \mathcal{C} \cap \mathcal{Q}} \mathbb{E}\left[\left|\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j\right|^2\right]\right)$$

where $\{\mathrm{AG}_w\}_j = \mathrm{AR}\left(\{G_i(w)\}_j;\ i \in \mathcal{Q}\right)$. The robustness coefficient $\Delta(\rho, f)$ is a bounded positive real value that depends on ρ and f.

Several existing aggregation rules satisfy the above definition. For example, we show in Proposition 1 that *Trimmed Mean* [43] satisfies Definition 1. We now formally define the trimmed mean function $\textsc{Tm} : \mathbb{R}^m \longrightarrow \mathbb{R}$ as follows: let $u_1, \ldots, u_m$ be a random sample from a given distribution, and let $u_{(k)}$ be the k^{th} order statistic of the sample. Then:

$$\textsc{Tm}(u_1, \ldots, u_m) = \frac{1}{m - 2t}\sum_{i=t}^{m-t} u_{(i)}$$

where $t \geq f$ is the truncation parameter.

Proposition 1. *Let $\rho > 2f$. Then, for all $w \in \mathbb{R}^d$ and $j \in [d]$, and for any $\mathcal{Q} \subset [n]$ such that $|\mathcal{Q}| > \rho$, we have*

$$\mathbb{E}\left[\left(\textsc{Tm}_j - \{\nabla \mathcal{L}(w)\}_j\right)^2\right] \leq \Delta(\rho, f)\left(\frac{1}{|\mathcal{C} \cap \mathcal{Q}|}\sum_{i \in \mathcal{C} \cap \mathcal{Q}} \mathbb{E}\left[\left|\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j\right|^2\right]\right)$$

where $\Delta(\rho, f) = \frac{2|\mathcal{C} \cap \mathcal{Q}|(f + |\mathcal{C} \cap \mathcal{Q}|)}{(|\mathcal{C} \cap \mathcal{Q}| - f)^2}$ and $\textsc{Tm}_j$ denotes $\textsc{Tm}\left\{(G_i(w)\}_j, i \in \mathcal{Q}\right)$.

In the following, we use *Trimmed Mean* as a default aggregation rule, tolerating up to half of the workers, acting as Byzantine, but any aggregation rule can be used instead. In fact, our theoretical analysis applies for all aggregation rules satisfying Definition 1, as soon as the robustness coefficient $\Delta(\rho, f)$ is sufficiently well behaved. Simply put, for a fixed number of Byzantine workers f, the aggregation error should at least be constant, if not decreasing, when the total number of workers ρ grows. This includes a large range of aggregation rules such as *Phocas* [40], *coordinate-wise median* [43] or *AKSEL* [7]. Hereafter, we assume that $\Delta(\rho, f)$ is indeed a function which is constant or decreases when ρ grows, as it is the case for Trimmed Mean (decreasing and asymptotically constant).

2.4 Assumptions

To formally analyze the convergence of our algorithm, we make the following standard assumption on the smoothness of the loss function [6]. We recall that $\{u\}_j$ denotes the j-th element of vector u.

Assumption 1 (Smoothness)**.** *For all $j \in [d]$, there exists $L_j < \infty$ such that, for all $w, w' \in \mathbb{R}^d$,* $|\{\nabla \mathcal{L}(w')\}_j - \{\nabla \mathcal{L}(w)\}_j| \leq L_j \left\|w' - w\right\|$.

Although most of our results do not rely on the convexity assumption, we also present a convergence result when the loss function is strongly convex, as stated below.

Assumption 2 (Strong convexity). *There exists $0 < \mu < \infty$ such that, for all $w, w' \in \mathbb{R}^d$, $\mathcal{L}(w') \geq \mathcal{L}(w) + \langle \nabla \mathcal{L}(w), w' - w \rangle + \frac{\mu}{2} \|w' - w\|^2$, where $\langle \cdot, \cdot \rangle$ denotes the inner product.*

To analyze the convergence of HGO under data heterogeneity, we also make the following two assumptions. Note that we do not rely on the (stronger) assumption of uniformly bounded variance, which is indeed quite difficult (and perhaps even impossible) to satisfy in the case of strong convexity [31]. Instead, we assume a non-uniform bound, as stated below in Assumption 4.

Assumption 3 (Bounded bias). *For all $i \in [n]$ and $j \in [d]$, there exist $\delta_{i,j} < \infty$ such that, for all $w \in \mathbb{R}^d$, $|\mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j] - \{\nabla \mathcal{L}(w)\}_j| \leq \delta_{i,j}$ where $x \sim \mathcal{S}_i$ denotes uniform sampling of x in $\mathcal{S}_i$.*

Assumption 4 (Non-uniform variance). *For all $w \in \mathbb{R}^d$, there exists $M_w < \infty$ and $Q_w < \infty$ such that, for all $i \in [n]$ and $j \in [d]$,$\mathbb{E}_{x \sim \mathcal{S}_i} \left[(\{\nabla \ell(w, x)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i} [\{\nabla \ell(w, x)\}_j])^2 \right] \leq M_w + Q_w \|\{\nabla \mathcal{L}(w)\}_j\|^2$.*

3 Heterogeneous gradient-based Optimization

We now present our protocol called **H**eterogeneous **g**radient-based **O**ptimization (HGO).

3.1 Overview of the protocol

The protocol, summarized in Algorithm 1, follows K steps of the following iterative procedure.

1. At each iteration $k \in [K]$, the server maintains a parameter vector w_k. To update its vector, the server chooses a random order of coordinate indices C_k (i.e., C_k is a permutation of the set $\{1, \ldots, d\}$), selects a random set of q workers, and broadcasts (w_k, C_k) to these workers. Then, the server waits for τ time units for workers' responses.

2. Upon receiving the broadcast message from the server, an honest worker i randomly samples a mini-batch of size s_i from its local data, and sends back to the server the first $b_i = \min\{\frac{\tau \lambda_i}{s_i}, d\}$ coordinates in C_k of its stochastic gradient $G_i(w_k)$, as defined in (2) above. Note that a faster worker (with higher computational capabilities) may compute more coordinates of its stochastic gradient compared to a slower worker. Byzantine workers may provide arbitrary values for their partial derivatives.

3. After τ time units, the server identifies the set of indices $\mathcal{B}_k \subseteq C_k$ that have been computed by at least v workers out of the q selected workers (the procedure for this is described in steps 8, 9 and 10 of Algorithm 1). The server then applies a robust *aggregation rule* AR on the consistent coordinates of the gradients sent by the workers, and uses its output to update the current parameter vector w_k (step 13 in Algorithm 1).

3.2 Time complexity.

The standard distributed SGD algorithm aggregates the full gradients computed by all n workers, while HGO only aggregates truncated gradients computed by $q < n$ random workers. Accordingly, HGO is faster than SGD when comparing these steps. Furthermore, while SGD's per-iteration time is dictated by the slowest worker, which means that the server waits for τ_s time units for the weakest worker to compute its gradient, HGO only waits for $\tau < \tau_s$ time units at each iteration, and the workers adjust to this time budget. However, note that the procedure of identifying the adequate block of coordinates and the workers that computed each of these coordinates (steps 8,9,10 and 11 in Algorithm 1) induces an overhead in computation when considering the total time complexity of the algorithm. Overall, the relative speed of HGO compared to SGD depends on the relation between $\tau - \tau_s$ and the overhead cost, which can be compared to $\mathcal{O}(q \log(q) \frac{\bar{\lambda} \tau}{\bar{s}})$, where $\bar{\lambda}$ is the average computation rate and $\bar{s}$ is the average mini-batch size of the workers.

Algorithm 1 **H**eterogeneous **g**radient-based **O**ptimization

Parameter Server

Input: q, AR (aggregation rule), w_1, v, K, γ
Execute the following instructions for each step $k = 1, \ldots, K$
$C_k \longleftarrow$ random permutation of $\{1, \ldots, d\}$
Select q random workers $\{i_1, \ldots, i_q\} \subseteq \{1, \ldots, n\}$
Broadcast (w_k, C_k) to the selected q workers
Wait τ time units to receive workers' gradients
$\nabla_{i_1}, \ldots, \nabla_{i_q} \longleftarrow$ partial derivatives sent by the q workers, i.e., elements of their gradients at w_k
$R \longleftarrow$ sort $\big(|\nabla_{i_1}|, \ldots, |\nabla_{i_q}|,$ decreasing order$\big)$
$b_v \longleftarrow v^{\text{th}}$ item in the list R
$\mathcal{B}_k \longleftarrow$ first b_v elements in C_k
$\forall j \in \mathcal{B}_k$, identify the set $\mathcal{Q}_j \subseteq \{i_1, \ldots, i_q\}$ of workers that sent the j-th coordinate of their gradients.
Compute the aggregate of workers' partial derivatives $\text{AG}_{w_k} \in \mathbb{R}^d$ such that, for all $j \in [d]$,

$$\{\text{AG}_{w_k}\}_j = \begin{cases} \text{AR}\left(\{\nabla_i\}_j,\ i \in \mathcal{Q}_j\right) & \text{if } j \in \mathcal{B}_k \\ 0 & \text{otherwise} \end{cases}$$

where $\{\nabla_i\}_j$ is sent by worker i for the j-th element of its gradient.
Update the parameter vector: $w_{k+1} = w_k - \gamma \text{AG}_{w_k}$

Honest Worker i

Input: local dataset $\mathcal{S}_i$, mini-batch size s_i, computational capability λ_i
If the broadcast message (w_k, C_k) is received **then** execute the following steps:
Sample a random mini-batch ζ_{i_k} of size s_i from dataset $\mathcal{S}_i$
$\mathcal{B}_k^i \longleftarrow$ first b_i elements in C_k, where $b_i = \min\{\frac{\tau\lambda_i}{s_i}, d\}$
Construct the set $\nabla_i = \big(\{G_i(w_k)\}_j, j \in \mathcal{B}_k^i\big)$, where $G_i(w_k)$ is as defined in (2)
Send back ∇_i to the Server

4 Theoretical Guarantees

Inspired by prior works in the homogeneous setting [5, 6, 7, 43], we show that our algorithm eventually reaches a ball centered at a local optimum, whose radius is a function of the statistical error. More formally, we show that, when $K \to \infty$, HGO eventually outputs a parameter estimate w such that

$$\mathbb{E}\left\|\nabla \mathcal{L}(w)\right\|^2 \leq \frac{d\Delta(v,f)U}{b\left(1 - \Delta(v,f)V\right)}$$

where $b = \liminf_{k \geq 1} \mathbb{E}[|\mathcal{B}_k|]$, $U = \max_{w \in \mathbb{R}^d} U_w, V = \max_{w \in \mathbb{R}^d} V_w$ with

$$U_w = \frac{1}{|C|} \sum_{j=1}^{d} \sum_{i \in \mathcal{C}} \left(\frac{M_w}{s_i'} + \delta_{i,j}^2 \right)$$

$$V_w = \frac{1}{|C|} \sqrt{\sum_{j=1}^{d} \left(\sum_{i \in \mathcal{C}} \frac{Q_w}{s_i'} \right)^2}$$

$$s_i' = \frac{s_i\left(|\mathcal{S}_i| - 1\right)}{|\mathcal{S}_i| - s_i}$$

Figure 2 illustrates the trajectory of the algorithm in the context where there exists a unique optimal solution w^*. In Theorem 1 and 2, we present the formal convergence result of our algorithm. In doing so, we divide the parameter space into two regions $\mathcal{W}$ and $\mathbb{R}^d \setminus \mathcal{W}$, where $\mathcal{W}$ is defined as:

$$\mathcal{W} := \left\{ w \;\middle|\; \min_{\mathcal{B} \subseteq [d]} \left\{ (1 - \Delta V_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - \Delta U_w \right\} > 0 \right\} \tag{4}$$

We denote by $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$ the partial gradient restricted to the coordinates in $\mathcal{B}$. Specifically, for all $x = (\{x\}_1, \ldots, \{x\}_d) \in \mathbb{R}^d$, $u = [x]_{\mathcal{B}} \in \mathbb{R}^d$ is such that $\{u\}_j = \{x\}_j$ if $j \in \mathcal{B}$, and 0 otherwise.

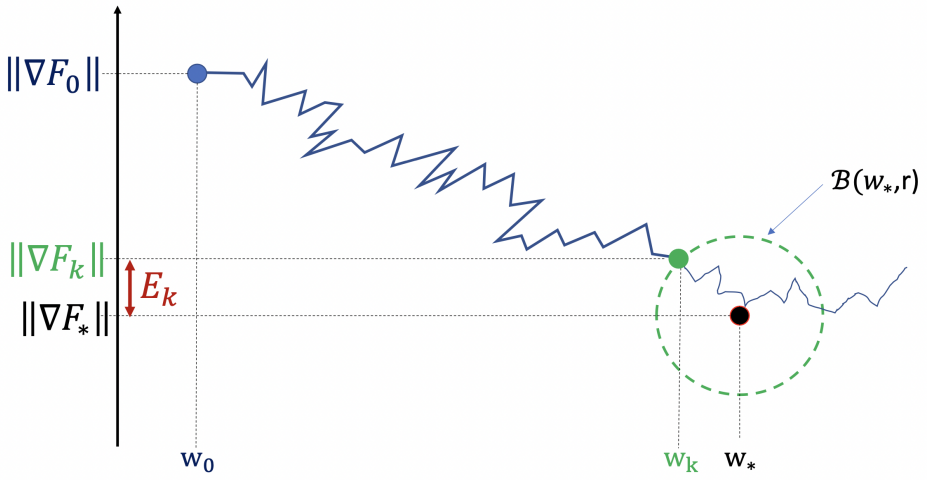


Figure 2: As long as the parameter vector w_k is inside $\mathcal{W}$, which means that the gradient norm is still big enough, the algorithm will keep approaching the ball $\mathcal{B}(w_*, r)$ centered at a local optimum, with a radius r (which is a function of the statistical error). When the condition is no longer satisfied, the perturbations induced by the presence of Byzantine workers will prevent progress towards the optimum. Then, the smaller r, the better the convergence guarantee of the algorithm in the presence of Byzantine faults.

The condition $(1 - \Delta V_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 > \Delta U_w$ can be seen as a generalization of the *variance-norm* condition first introduced in [5]. This work was considering the special case of SGD ($b = d$ and $\mathcal{B} = [d]$) with unbiased estimation ($\forall (i,j) \in [N] \times [d], \delta_{i,j} = 0$), uniform upper bound on the variance ($Q_w = 0$), and a fixed mini-batch of size 1 ($\forall i \in [n], s_i = 1$). In this special case, our condition reduces to the variance-norm condition, presented in Proposition 1 of [5]. The authors also claimed that their condition will be easier to satisfy when using larger mini-batch sizes. Our formulation confirms this claim: increasing the mini-batch size (s_i) increases the value of the right-hand side of the condition inequality and decreases the left-hand side one, making it easier to achieve a satisfying outcome in many problems.

4.1 Convergence analysis

Before presenting our main convergence results, note that, given a parameter vector w_k, it is hard to demonstrate progress towards a local optimum without first demonstrating that AG_{w_k} is pointing in the same direction as the actual gradient of the loss function. Specifically, the scalar product between the aggregated gradient AG_{w_k} and $\nabla \mathcal{L}(w_k)$ must be strictly positive. We show below in Lemma 1 that this condition holds true in our setting whenever $w_k \in \mathcal{W}$ and the aggregation rule AR we use is $\Delta(v, f)$-robust.

Lemma 1. *Suppose that Assumptions 3 and 4 hold true. Consider the k-th step of Algorithm 1. If* AR *is $\Delta(v, f)$-robust at w_k and $w_k \in \mathcal{W}$, then, denoting $\Delta = \Delta(v, f)$, we have:*

$$\mathbb{E}\left[\langle \mathrm{AG}_{w_k}, [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle\right] > \frac{1}{2}\left((1-\Delta V_{w_k})\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - \Delta U_{w_k}\right)$$

Using this result, we obtain the convergence properties of HGO for strongly convex and non-convex functions. Under Assumption 1, we define $L = \max_{i\in[d]} L_i$.

Theorem 1 (Strongly convex)**.** *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate γ. If $\gamma < \frac{1}{\sqrt{d}L}$ and $\Delta V < 1$, then*

$$\mathbb{E}\left[\mathcal{L}(w_K) - \mathcal{L}^*\right] \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta V)\right)^{K-1}(\mathcal{L}(w_1) - \mathcal{L}^* - H) + H$$

where $\mathcal{L}^*$ denotes the minimum value of $\mathcal{L}$ on $\mathbb{R}^d$, $b = \min_{k\in[K]} \mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right], U = \max_{k\in[K]} U_{w_k}, V = \max_{k\in[K]} V_{w_k}$, and $H = \frac{d\Delta U}{2b\mu(1-\Delta V)}$.

Unlike convex functions, non-convex functions may have multiple local minima, making convergence analysis more complex [6], especially for coordinate descent methods [28, 38], which are closer to our algorithm. While it is hard in general to upper-bound the optimality gap as in previous theorem, we upper-bound the average (and thus the minimum) gradient norm of the cost function, evaluated using the sequence of parameter vectors $\{w_k\}_{k\in[K]}$ generated through the iterations $1, \dots K$.

Theorem 2 (Non-convex)**.** *Suppose that Assumptions 1, 3 and 4 hold. If $\gamma < \frac{1}{\sqrt{d}L}$ and $\Delta V < 1$ then*

$$\min_{k\in[K]} \mathbb{E}\left[\left\|\nabla \mathcal{L}(w_k)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Kb\gamma(1-\Delta V)} + \frac{d\Delta U}{b(1-\Delta(v,f)V)}$$

where $U = \max_{k\in[K]} U_{w_k}$, $V = \max_{k\in[K]} V_{w_k}$ and $b = \min_{k\in[K]} \mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right]$.

4.2 Some remarks on the convergence result

On the learning rate. In the above theorems, we used a tight (safe) upper bound on the learning rate $\gamma < \frac{1}{\sqrt{d}L}$ to eliminate the dependence on the random block $\mathcal{B}$ constructed by the server at step 10 in Algorithm 1. However, HGO only requires the following condition $\gamma_k < \frac{1}{\sqrt{|\mathcal{B}_k|}L}$ at each iteration. Note that the new upper bound is looser and allows for big steps, depending on the block size $|\mathcal{B}_k|$, which varies at each iteration. In practice, one can use a dynamic learning rate γ_k, inversely proportional to $\sqrt{|\mathcal{B}_k|}$, to improve the convergence rate.

On the generality of our results. Our work encapsulates a large family of Byzantine-resilient and non-Byzantine-resilient optimization algorithms. For instance, in [38], the author studies Random Coordinate Descent (RCD), where a single random index ($b = 1$) is selected at each iteration, and the adequate coordinate is updated following a gradient step. In this simple algorithm, the gradient step is computed using a learning rate and a partial derivative, which is itself computed using the complete dataset. Since an exact non-distributed computation is executed, there will be no variance ($M_k = Q_k = 0$), no aggregation ($\Delta(v, f) = 0$), no mini-batch ($\forall i \in [n], s_i = 1$) and no bias ($\forall (i, j) \in [N] \times [d], \delta_{i,j} = 0$), which means that $U_w = V_w = 0$ and $H_k = 0$. Replacing these values in Theorem 1 recreates the convergence result for RCD, presented in the second part of Theorem 1 in [38].

4.3 Trade-offs

Our theoretical results highlight some interplay between the convergence speed, the computation power of the workers, as well as the dataset partitioning and the presence of Byzantine workers. We discuss some of them below.

Convergence vs time complexity. The robustness coefficient $\Delta(v, f)$ is a key factor of the aggregation error. Its value depends on the quality of the aggregation rule we use, but also on the number of Byzantine workers within the selection of random workers. For a fixed number f of Byzantine devices, increasing the number of devices sampled at every round q also increases the number of correct workers amongst the quorum of size v we use for the aggregation. This has a direct impact on the convergence rate as well as on the statistical error. As a matter of fact, one can see in both theorems that the first term of the right-hand side inequality decreases when $\Delta(v, f)$ decreases. Recall from Section 2.3 that $\Delta(v, f)$ is decreasing with respect to v. This simply means that a bigger sample size q implies a faster convergence rate. The statistical error is also positively correlated to $\Delta(v, f)$; hence, also to q. However, note that, as explained in Section 3.2, the time complexity of the algorithm also increases with q. In short, a better theoretical convergence implies a bigger time complexity.

Heterogeneity vs convergence. From our theoretical results, we can observe that increasing the mini-batch size s_i of the workers improves the statistical error as well as the convergence rate. On the other hand, these two convergence properties are also linked to the size $|\mathcal{B}_k|$ of the block $\mathcal{B}_k$ constructed by the server at iteration k. In fact, increasing $b = \min_{k\in[K]} \mathbb{E}_{\mathcal{B}_k}[|\mathcal{B}_k|]$ also increases the convergence rate and reduces the statistical error. To increase b we can either:

1. Decrease the minimum number of workers v to execute the aggregation. However, this quantity is lower-bounded by a function of the ratio of Byzantine workers the algorithm can tolerate; hence, it cannot be decreased arbitrarily without impeding the Byzantine resilience of the algorithm.

2. Decrease the mini-batch sizes, so that every worker dedicates its resources to computing more blocks ($\lambda_i \tau = b_i s_i$). However, note that it would induce a new trade-off, as diminishing the mini-batch sizes have a negative effect on the convergence of the algorithm.

3. The server can try to re-weight the worker sampling to contact the powerful workers more often (the ones with large computational capabilities λ_i), hence mechanically increasing the block size and the mini-batch size together. This would improve the convergence rate, but may exclude workers with valuable data from the learning procedure, which certainly leads to a worse statistical error when data are heterogeneous.

Byzantine tolerance vs hardware heterogeneity. Maybe the most interesting takeaway we can extract from our results is the interplay between hardware heterogeneity and Byzantine tolerance of the algorithm. In both results, we introduce a sufficient condition for convergence, namely $\Delta(v, f)V < 1$. This inequality formalizes the interplay between the Byzantine tolerance (through the robustness coefficient $\Delta(v, f)$) and the computational power of the workers (captured in the variable V). To tolerate a bigger Byzantine impact, V must decrease in order to follow a gradient step at round k. Decreasing V implies to increase the size of the mini-batches selected by the workers, to reduce the variance. This can be achieved by instructing the workers to favor the mini-batch size over the number of coordinates, when allocating the available computation resources. However, when the computational capabilities are not high enough, decreasing the number of coordinates leads to a small block size b, which results in poor convergence properties. Conversely, if each worker has strong capabilities, it will be possible to increase both b and s_i, in order to satisfy the condition and improve the convergence properties at the same time. In other words, the more weak devices are present in the network, the less the algorithm can tolerate Byzantine behaviors.

5 Empirical Evaluation

To demonstrate the practical relevance of our optimization scheme in a Byzantine-prone environment with heterogeneous participants, we ran an exhaustive set of experiments to evaluate the performance of HGO on several classification tasks. The main objective of this section is to demonstrate that our protocol improves the participation of slow devices, and outperforms traditional schemes such as SGD (in terms of final accuracy of the model); hence highlighting the importance of our method when learning with heterogeneous and Byzantine participants. To do so, we first present a set of experiments on simple models such multinomial

logistic regression. Then we propose an extension of HGO to more complicated models, such as feed-forward neural networks, and discuss some future direction for our protocol in the context of convolutional neural networks. To simplify the presentation, we evaluate the performance of HGO against SGD to eliminate the impact of more advanced solvers such as NAG [29], RMSprop [16], AdaGrad [12] or Adam [19] that are still misunderstood from a technical viewpoint.

Due to page limitation, we only present a few experiments that support our theoretical analysis. A more complete set of experiments involving more ML models and datasets, is available in the extended version of this paper. An Android implementation of our algorithm is also available to execute it on real smartphones. The source code of our experiments and the Android implementation are available in the following GitHub repositories, respectively: `https://anonymous.4open.science/r/HgO-5136` and `https://anonymous.4open.science/r/HgOApp-CC8E`.

5.1 Experimental setup

Models and datasets. We evaluated our procedure on a broad class of models inclusing linear regression, binary logistic regression and support vector machine on several benchmark datasets such as Boston[2], Wisconsin Breast Cancer[3], or Phishing[4]) to confirm the theoretical results. But we also considered more complex setups, namely: multinomial logistic regression and feed-forward neural network trained on MNIST[5] and Fashion-MNIST[6]. We only present in the main paper a multinomial logistic regression and a feed-forward neural network trained on the MNIST dataset, but our findings were consistent across the set of experiments we considered.

Note that, for a fair comparison, we use the same learning rate and mini-batch size for SGD and HGO in every experiments.

Network architecture and hardware specification. We implement the parameter-server architecture where a trusted server communicates through message passing with a set of workers, to learn the model. To distinguish the hardware specifications of workers, we classify them in three categories characterizing their computational capabilities, namely: "weak", "average" and "powerful". Each category represents a set of workers whose computation rates are normally distributed over a fixed mean value that depends on the learning task. Accordingly, we consider varying computation profile for the network:

- $C_{standard}$: 30% of weak devices, 40% average devices and 30% powerful devices
- C_{weak}: all devices are weak
- C_1: 90% of weak and 10% of average devices
- C_2: 70% of weak and 30% of average devices
- $C_{powerful}$ all devices are powerful

When the computation profile of the network is not specified, we assume that it is the standard scenario $C_{standard}$.

At each iteration, the server only contacts a random set of q workers, and aggregates their responses using a given aggregation rule. By default, we set $\frac{q}{n} = 0.8$, and use *Average* and *Median* (a special case of Trimmed Mean) as aggregation rules in honest and Byzantine environments, respectively.

Data heterogeneity. We experiment both homogeneous and heterogeneous partitioning of datasets over the workers. In the homogeneous case, each worker is assigned a non-overlapping independent and identically distributed sample from the training set. We construct heterogeneous versions of the datasets by

[2] `http://lib.stat.cmu.edu/datasets/boston`
[3] `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)`
[4] `https://archive.ics.uci.edu/ml/datasets/phishing+websites`
[5] `http://yann.lecun.com/exdb/mnist/`
[6] `https://github.com/zalandoresearch/fashion-mnist`

restricting the classes detained by each workers. Specifically, in our experiments, each worker only has access to data points for two classes out of ten from the MNIST dataset.

Byzantine workers. We use two state-of-the-art Byzantine attacks [3, 41] to evaluate the impact of our new learning scheme on the robustness of existing aggregation rules (Krum [5], Trimmed Mean [40] and Aksel [7]). These attacks exploit the normal distribution of correct estimates to construct Byzantine values that are still within a few standard deviations from the true mean. Byzantine workers start implementing the aforementioned attacks at iteration 10 of each experiment. As a benchmark, we always plot the test accuracy of averaging under no attack.

5.2 Impact of waiting time on accuracy and runtime

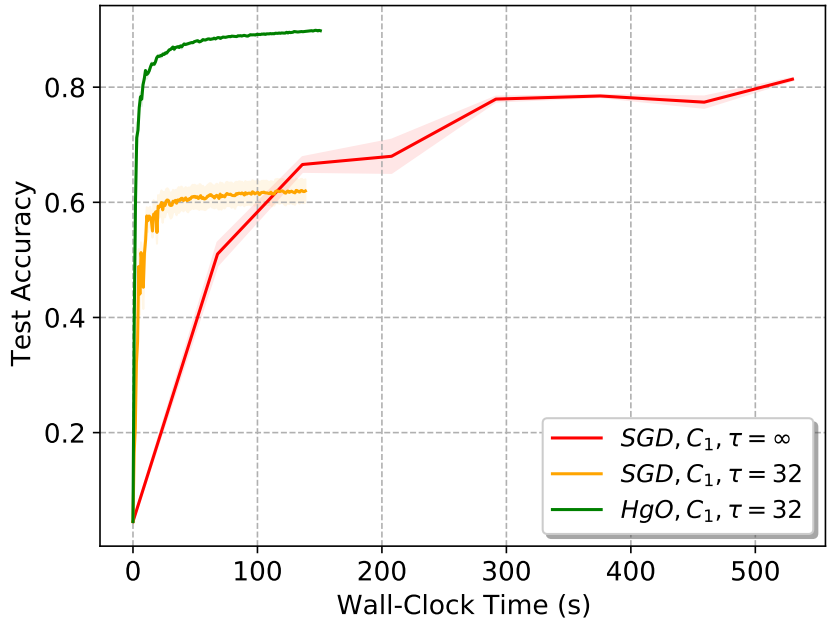
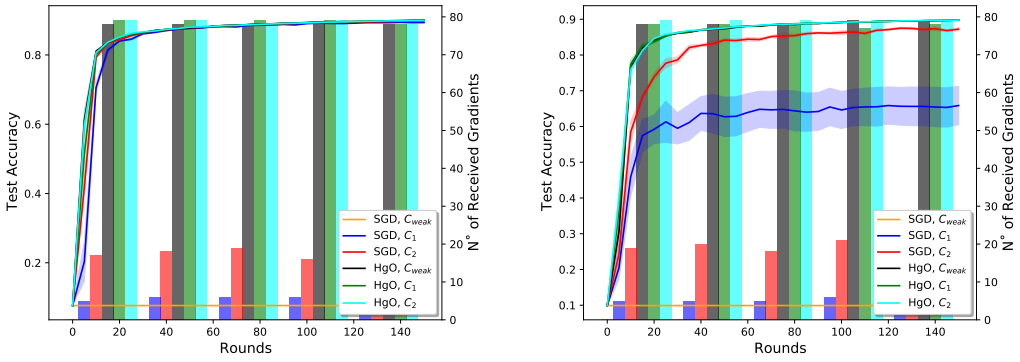


Figure 3: Training a multinomial logistic regression on MNIST, heterogeneously partitioned over $n = 100$ honest workers. The server averages the estimates of $q = 80$ random workers at each iteration. When $\tau = \infty$, the server waits for the slowest device to send its message. When $\tau = 32$, the server waits for 32 time units and only aggregates the received messages.

In Figure 3, we compare the accuracy of a multinomial regression on MNIST trained with SGD or HgO against the runtime of the protocols.[7] When we do not force any limit on the waiting time of the server ($\tau = \infty$), it must wait for every worker to complete its computing task before executing any aggregation. Consequently, the algorithm speed of SGD is limited by the hardware specifications of the weakest device of the network. While this approach is indeed slow, it allows the model to learn from all the local datasets of the workers, which results in a good accuracy. A practitioner might be tempted to reduce the waiting time by fixing a time limit for each iteration (e.g. $\tau = 32$ time units). By doing so, we can observe a substantial speedup because the server is discarding the workers that cannot provide a response within the time budget. However, when the dataset is not homogeneously partitioned, The model learned with SGD will miss the data points held by the slow workers; hence resulting in poor final accuracy.

[7] To measure this runtime, we us the actual wall-clock time, which gives a clear idea of the runtime as opposed to the number of algorithm step which simply give an idea on the convergence rate of the protocols.



(a) Homogeneous dataset partitioning

(b) Heterogeneous dataset partitioning

Figure 4: Training a multinomial logistic regression on MNIST, using $n = 100$ workers in an honest setting. HGO shows improved convergence properties (final accuracy, convergence rate) w.r.t. SGD, even in the weakest scenario C_{weak} where all workers have low hardware profiles.

With HGO, even when limiting the server waiting time, the workers can adjust their computing task to the time budget, by computing a few partial derivatives using a few data points in the mini-batch. That way, the model is visiting the full dataset either through full gradients (computed by powerful workers) or partial derivatives (computed by weaker workers). This results in a much better overall accucary of the model without slowing down the overall execution of the procedure. In the next experiments, we always set a waiting time τ for HGO and SGD equal to 32.

5.3 Impact of heterogeneous partitioning

As we explained above, the principal advantage of HGO is allowing the server to receive information from all the workers, at each iteration. Note that when the dataset is homogeneously partitioned over the workers, it is sufficient to have a few workers in order to get a representative sampling of the full dataset. This situation is illustrated in Figure 4a, where SGD is able to reach a good accuracy, even when not all the workers are participating in the training process, except for the weakest scenario C_{weak} where no worker is able to compute a full gradient. However, in the heterogeneous case illustrated in Figure 4b, HGO clearly outperforms SGD as discarding weaker workers now comes at the expense of losing valuable information from their local datasets. While SGD's performance is strongly correlated to the computation capabilities of the workers, HGO shows roughly the same convergence properties, independently of the hardware profile of the workers. Interestingly, using HGO allows to reach top performance, even with a network full of weak devices, which is impossible with standard learning schemes.

5.4 Improving weak networks with a few average devices

In the previous experiments (Figure 4), one can see that HGO shows nearly the same convergence pattern for all scenarios. In the experiment reported in Figure 5, we voluntarily set the computation rates of the weak devices to very low values, in order to get a lower accuracy using the weakest scenario C_{weak}. While using HGO with a network full of weak workers still shows good performance, we demonstrate that replacing only 10% of these weak devices by average workers is sufficient to match the convergence quality of an all powerful network on MNIST. This experiment is very interesting and motivates the use of HGO in industry-scale machine learning applications. For instance, a company may train a complex machine learning model using HGO, with the data available in smartphones, which can be considered as weak devices. Then, it can introduce a few powerful computers to boost the learning performance. The result would be similar to a scenario where all the data is relocated into one location and trained with strong computers using SGD.

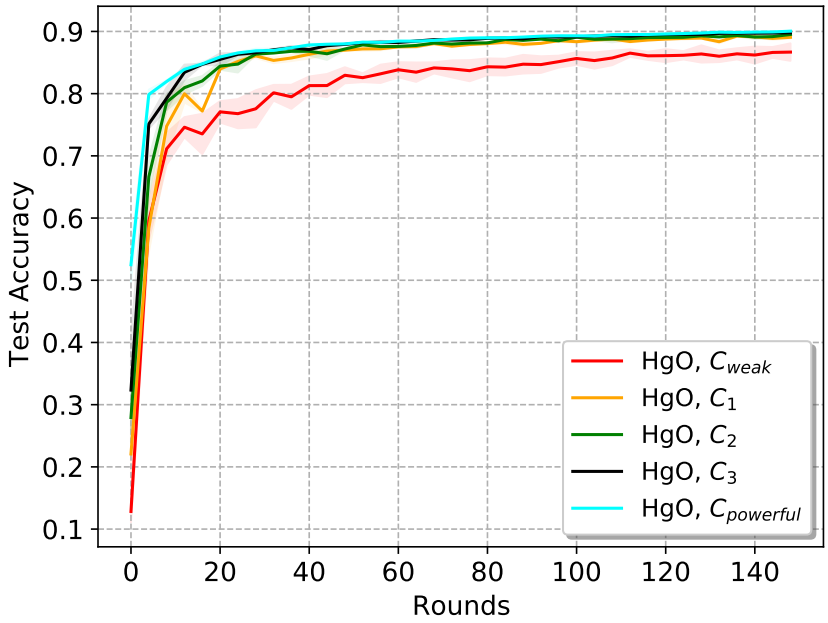


Figure 5: Training a multinomial logistic regression on MNIST, homogeneously partitioned over $n = 100$ honest workers. The server averages the estimates of $q = 80$ random workers at each iteration. Interestingly, replacing 10% of weak devices by average ones (C_1) is sufficient to match the performance of ($C_{powerful}$).

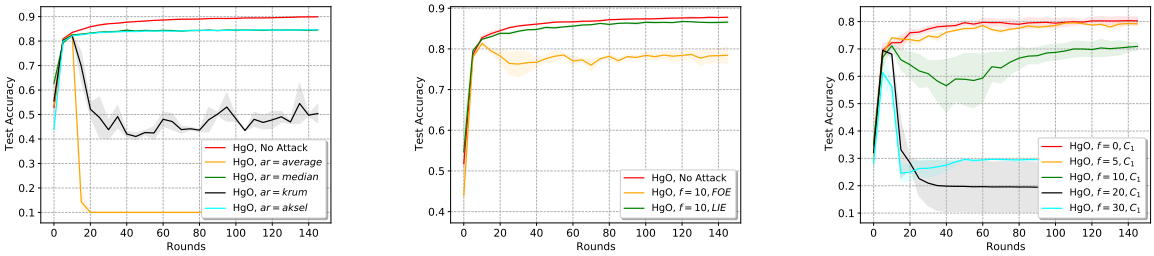
5.5 Analyzing the impact of $\Delta(v, f)$

Another interesting takeaway of our analysis is the interplay between the strength of Byzantine workers and the computational heterogeneity, represented by the condition $\Delta(v, f)V < 1$. This condition has a direct impact on the convergence rate, as well as on the statistical error of the procedure, and can be influenced in many ways. We discuss three of them below.

Performance of aggregation rules. Aggregation rules do not perform the same way against Byzantine workers. In Figure 6a, the simplest aggregation function, which is *Average*, do not withstand a single Byzantine worker. Therefore, HgO combined with averaging is not converging. On the other hand, *Krum*, *Median* and *Aksel* are indeed tolerating a fraction of Byzantine workers, but the Byzantine impact is stronger on *Krum* than the others, which are approaching the convergence properties of averaging under no attack.

Attack strategies. Byzantine workers may collude and implement complex strategies to make the learning ineffective. In figure Figure 6b, the server is using *Median* as an aggregation rule to defend against $f = 10$ Byzantine workers. We can see that the impact on the attack "FOE" [41] is more devastating the the impact of "LIE"[3].

Number of Byzantine worker. Finally, the number of Byzantine workers is naturally a strong variable when it comes to robust aggregation. In Figure 6c, we increase the number of Byzantine workers from 0 to 30 and we can see that poor convergence properties are linked to high number of Byzantine workers. In this experiment, we voluntarily use a heterogeneous partitioning of the dataset and a relatively weak network C_1 to make sure that the variable V has a high value and high values of Byzantine workers $f = 20, f = 30$ to increase the value of $\Delta(v, f)$. Then HgO is arguably not converging because the condition $\Delta(v, f)V < 1$ is broken.



(a) Homogeneous dataset partitioning

(b) Homogeneous dataset partitioning

(c) Heterogeneous dataset partitioning

Figure 6: Training a multinomial logistic regression on MNIST, with a total of $n = 100$ workers. The server aggregates the partial gradients of $q = 80$ random workers at each iteration. By default, we use Median as an aggregation rule and set $f = 10$. In each experiment, we vary the number of Byzantine workers, their attack and the aggregation rule used, which are all factors impacting the variable Δ, leading to different convergence properties. In the last experiment (c), we construct an extreme scenario where the number of Byzantine workers is increased to get closer to the number of correct ones, using a heterogeneous dataset partitioning, to intentionally dissatisfy the condition $\Delta V < 1$.

5.6 Extension to neural networks

Note that simple linear models have separable cost functions, which means that computing partial derivatives can be done independently. However, for more complex models such as neural networks, computing the gradient relies on the back-propagation algorithm and the chain rule. This makes the computation of a partial derivative much more dependent on the model architecture than in the linear case. Most importantly, picking a block of random coordinates from the full gradient will result in an unbalanced updates between the network layers. In this section, we present a variant of HGO, where workers only update sub-network at each round of the learning.

Adaptation of HGO to Feed-forward networks. Let us consider a feed-forward network with M layers. To compute a partial gradient at each round of the protocol, each worker samples a subset of the network weights' on the $M - 1$ first layers of the network; hence building a sub-network. The output layer should be kept unchanged for consistency purposes (the model should still be able to classify within the same number of classes). Naturally, the number of sampled weights in each layer depends upon the computational capabilities of the worker. Then each worker computes their gradient on the obtained sub-network to mimic the computation of a partial gradient at each round. This procedure is illustrated in Figure 7 for a simple 3 layers feed-forward network.

The gradient of the sub-network (whose dimension is less than the original gradient) can be considered as a partial gradient, or a block of random gradient coordinates, covering all the layers of the neural network. For simpler models, the random block is a vector and the block size is an integer. However, for feed-forward neural networks, a block is a tensor and its size is a vector of integers. For example, feed-forward neural network with input size 784, one hidden layer with 30 neurons and an output layer with 10 classes, a block size will be written in the following format: $[98, 20, 10]$. This means that 98 out of 784 weights in the first layer and 20 out of 30 weights in the second layer will be sampled and updated at each round (as explained above, the last layer remains unchanged).

Relevance of HGO for feed-forward networks. Using the new approach for block generation, we demonstrate in Figure 8 that the practical relevance of HGO can be extended to feed-forward neural networks.

Recall that, when $\tau = \infty$, SGD is indeed using the full dataset partitioned over the workers. However, the server must wait for the slowest devices to compute its gradient estimate, which degrades the runtime. To accelerate the process, the server sets a limit on the waiting time $\tau = 32$. By doing so, some workers will not be able to compute their gradient in time and will be discarded from the training, which results in a poor accuracy. Even in the context of feed-forward networks, our varient of HGO combines the runtime efficiency and accuracy.

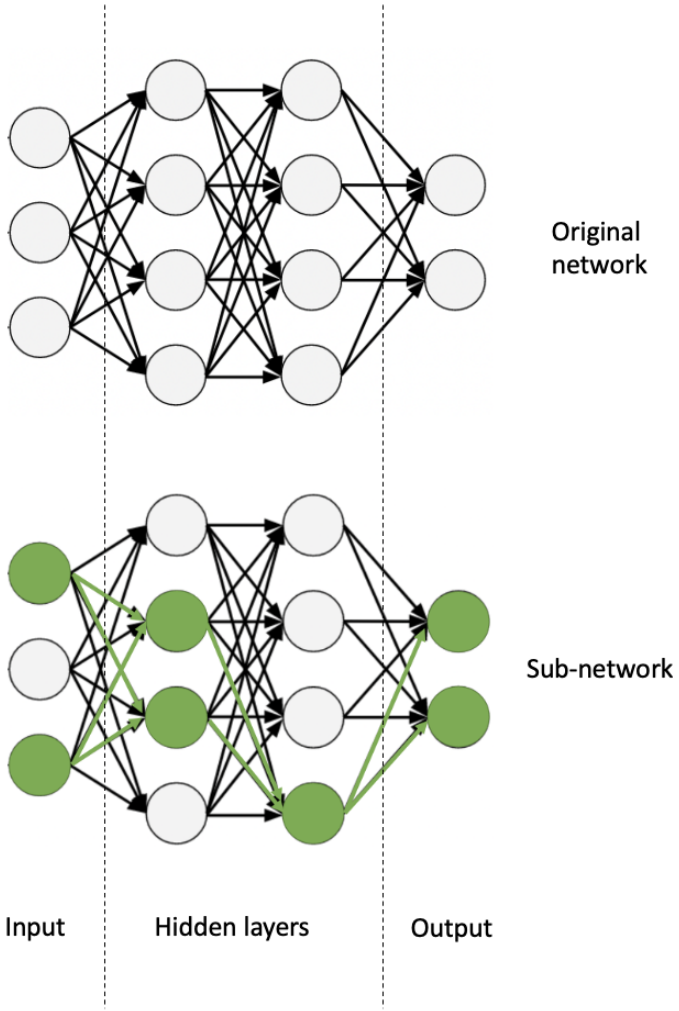
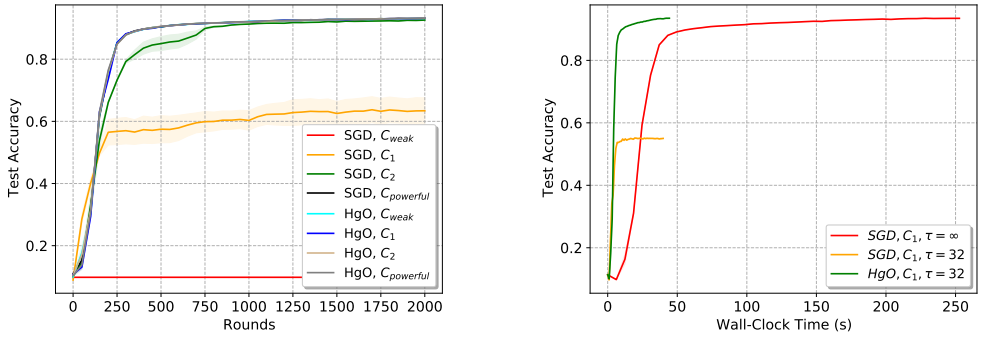


Figure 7: Constructing a sub-network from a fully connected feed-forward neural network.

Note that this variant approach to HGO can be linked to the dropout technique [34], widely used among ML practitioners. When employing dropout in a neural network with classical learning schemes, a fraction of weights are ignored in the training phase, with some probability. In HGO, each worker will dropout a fraction of the weights, depending on its computing capabilities, in a deterministic fashion. Powerful devices can use the full neural network without applying the dropout. As regularization techniques, the dropout approach prevents over-fitting, and this might be another explanation of the better final test accuracy of HGO over SGD, besides the fact that HGO includes all the local datasets in the training.

More advanced architectures like convolutional neural networks (CNN) need a different way of constructing a sub-network. In fact, the classification phase of a CNN is a simple feed-forward network, on which we can easily applied the sub-network technique. However, the feature extraction phase (convolutions and pooling) is challenging. In this work, we limit our extension to feed-forward neural networks and leave the CNN extension for future work.



(a) (b)

Figure 8: Training a feed-forward neural network (fully connected network with 784 inputs, one hidden layer of 30 neurons and 10 outputs) on a heterogeneously partitioned MNIST dataset, with a total of $n = 100$ honest workers. Depending on its hardware capability, each device uses a random sub-network to compute the gradient estimate. Powerful devices use the full neural network.

6 Related Work

Prior works in distributed ML consider the issues of heterogeneous computational capabilities, heterogeneous datasets and Byzantine behavior separately. Several approaches studied the Byzantine resilience of gradient-based descent optimization algorithms [2, 5, 7, 8, 9, 10, 42, 43] by leveraging robust statistics, filtering schemes or coding theory, to defend against a fraction of Byzantine workers. However, they all assumed unbiased estimation and i.i.d. local datasets (homogeneous distribution of data).

Some other works [18, 20, 23] studied SGD with local iterations on heterogeneous data, without assuming Byzantine workers. Distributed SGD was also analyzed in [11, 32], assuming data heterogeneity and Byzantine attacks. However, the use of encoding schemes in this literature makes the approach not practical. In [14], the authors combined a clustering scheme with robust statistics to transform a heterogeneous dataset into homogeneous clusters before any distributed optimization step. Besides the fact that the proposed scheme uses an extra step (clustering) to handle heterogeneity, which impacts time complexity, reassigning data to clusters rises privacy concerns. In all these approaches, the modeling of heterogeneity considered a dissimilarity at the optimum between the local loss functions, gradients or parameter vectors. Most importantly, none of these works considered the heterogeneity of workers in terms of computational capabilities.

Note also that, while compression schemes [4, 25, 35, 37] were proven efficient in reducing communication complexity, they do not address computational heterogeneity. Moreover, prior works only considered computational heterogeneity through the window of asynchrony [9, 17, 24], rather than adapting the algorithm itself to the computational capabilities of workers. Particularly, the effect of stragglers in a heterogeneous network of workers was handled in [17] using a smart learning rate schedule. Yet, the workers still use the same optimization algorithm (SGD). Besides, Byzantine resilience and data heterogeneity were not considered.

Our protocol provably converges under a large set of constraints: Byzantine faults, stale and non-fully active workers, data heterogeneity and hardware heterogeneity. In fact, we model all kinds of inaccuracies at the worker level (of which heterogeneity is an example) using a coordinate-wise bias on partial derivatives at each iteration, which is interesting in its own right and actually different from assuming dissimilarity at the optimum.

7 Conclusion

We presented HGO, a Byzanzine-resilient distributed learning scheme whose iteration cost can be adapted to the capabilities of workers (e.g., smartphones). HGO encourages the participation of slower devices,

improving the accuracy of the model when the participants do not share the same dataset. Our protocol also tolerates Byzantine devices, by combining a randomized sampling of devices with a robust aggregation scheme. We proved HGO's convergence under a non-trivial combination of hypotheses, namely: Byzantine failures, computational heterogeneity and data heterogeneity. We also demonstrated important tensions between relevant quantities such as the statistical error rate, the impact of Byzantine workers, the level of inaccuracy of honest workers, the time complexity of the algorithm and its convergence rate. We support our theoretical analysis empirically, by demonstrating the interplay between convergence properties and the execution environment of the algorithm.

Interestingly, we show that introducing as few as 10% of average devices in a network full of weak devices is sufficient to reach the performance of a network with only powerful devices. HGO clearly outperforms SGD in the case of non-identically distributed datasets by aggregating the estimates of all workers, regardless of their computational capabilities. This contrasts with SGD, which imposes the same workload on all workers, and therefore discards some workers (which data may be very valuable to the training). On a more ethical side, our work encourages data privacy by making smartphones (which are the most important personal data carrier nowadays) contribute directly to AI projects, without exposing local data to third parties.

To focus on HGO's originality, we assumed a unique and trusted parameter-server [1, 21], a common assumption in Byzantine machine learning [2, 5, 7, 8, 9, 10, 42, 43]. However, HGO can be used with the replication scheme of [13] to circumvent that assumption. Furthermore, this work opens an interesting line of open questions, among which: 1) Can this work be extended to second-order optimization and to convolutional neural networks? 2) Which computation power profile is needed to reach a target accuracy within a given time budget? And finally 3) what is the impact of network bandwidth on the convergence properties? We leave these questions open for future works.

References

[1] Martin Abadi et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 4613–4623. Curran Associates, Inc., 2018.

[3] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8635–8645. Curran Associates, Inc., 2019.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems, 2018.

[5] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. NeurIPS'17, page 118–128, Red Hook, NY, USA, 2017. Curran Associates Inc.

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

[7] Amine Boussetta, El-Mahdi El-Mhamdi, Rachid Guerraoui, Alexandre Maurer, and Sébastien Rouault. AKSEL: Fast Byzantine SGD. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[8] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients, 2018.

[9] Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Rhicheek Patra, and Mahsa Taziki. Asynchronous Byzantine machine learning (the case of SGD). 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 1145–1154, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[10] D. Data and S. Diggavi. Byzantine-tolerant distributed coordinate descent. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2724–2728, 2019.

[11] Deepesh Data, Linqi Song, and Suhas Diggavi. Data encoding methods for byzantine-resilient distributed optimization. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2719–2723, 2019.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[13] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê Nguyên Hoang, and Sébastien Rouault. Genuinely distributed byzantine machine learning. page 355–364, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment, 2019.

[15] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. SGD: General analysis and improved rates. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5200–5209. PMLR, 09–15 Jun 2019.

[16] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[17] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 463–478, New York, NY, USA, 2017. Association for Computing Machinery.

[18] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtarik. Tighter theory for local sgd on identical and heterogeneous data. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4519–4529. PMLR, 26–28 Aug 2020.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[20] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian U. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *ICML*, pages 5381–5393, 2020.

[21] Jakub Konecný, H. McMahan, D. Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *ArXiv*, abs/1610.02527, 2016.

[22] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[23] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020.

[24] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for non convex optimization. *In NIPS, pages 2737–2745*, 2015.

[25] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

[26] Ji Liu, Stephen J. Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *J. Mach. Learn. Res.*, 16(1):285–322, jan 2015.

[27] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[28] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[29] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[30] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014.

[31] Lam Nguyen, Phuong HA Nguyen, Marten van Dijk, Peter Richtarik, Katya Scheinberg, and Martin Takac. SGD and hogwild! Convergence without the bounded gradients assumption. volume 80 of *Proceedings of Machine Learning Research*, pages 3750–3758, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[32] Shashank Rajput, Hongyi Wang, Zachary B. Charles, and Dimitris Papailiopoulos. Detox: A redundancy-based framework for faster and more robust gradient aggregation. In *NeurIPS*, 2019.

[33] Peter Richtárik and Martin Takáăź. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Math. Program.*, 144(1–2):1–38, apr 2014.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[35] Haobo Sun, Yingxia Shao, Jiawei Jiang, Bin Cui, Kai Lei, Yu Xu, and Jiang Wang. Sparse gradient compression for distributed sgd. In Guoliang Li, Jun Yang, Joao Gama, Juggapong Natwichai, and Yongxin Tong, editors, *Database Systems for Advanced Applications*, pages 139–155, Cham, 2019. Springer International Publishing.

[36] Qing Tao, Kang Kong, Dejun Chu, and Gaowei Wu. Stochastic coordinate descent methods for regularized smooth and nonsmooth losses. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 537–552, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[37] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[38] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[39] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd, 2018.

[40] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent, 2018.

[41] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation, 2019.

[42] Z. Yang and W. U. Bajwa. Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning. *IEEE Transactions on Signal and Information Processing over Networks*, 5(4):611–627, 2019.

[43] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2018.

A Preliminaries

A.1 Notations

We recall all the variables used in this work and their corresponding designation.

Table 1: List of variables and their designation

N	Total number of data points across the network
n	Number of workers
λ_i	Computation rate of worker i
f	Maximum number of Byzantine workers
$\mathcal{Q}_j$	Set of workers that computed coordinate j
$\mathcal{C}_j$	Set of correct workers that computed coordinate j
q	Number of random workers selected by the server
τ	Number of time units the server waits for worker's estimations
AR	Robust aggregation rule
Δ	Robustness coefficient of AR
v	Minimum number of workers to execute the aggregation
μ	Strong convexity parameter
L_i	Coordinate-wise lipschitz parameter
w_k	Parameter vector at iteration k
d	Dimension of the parameter vector
$\mathcal{S}_i$	Local data set of worker i
ζ_i	Random mini-batch selected by worker i
s_i	$= \lvert\zeta_i\rvert$ mini-batch size
$\mathcal{B}_i$	Block of coordinates computed by worker i
b_i	$= \lvert\mathcal{B}_i\rvert$ Block size
$\mathcal{L}$	Cost function being optimized
$l(w, x)$	loss computed at w using the data point x
G_i	Gradient estimation of worker i
$\nabla \mathcal{L}(w)$	True gradient
$\{X\}_j$	j^{th} coordinate of vector X
$[X]_{\mathcal{B}}$	$\{[X]_{\mathcal{B}}\}_j = \{X\}_j$ if $j \in \mathcal{B}$, $= 0$ otherwise
γ	Learning rate (step size)
$\delta_{i,j}$	Bias parameter of worker i at coordinate j
M_k, Q_k	Variance parameters of $[\nabla l(w, x)]_j, \forall j \in [d]$ and $w \in \mathbb{R}^d$
F_0	$= \mathcal{L}(w_1) - \mathcal{L}(w_*)$ (initial gap)

B Proof of theoretical results

B.1 Proposition 1

Trimmed Mean (TM) is a classical robust statistic that has been studied in the literature, especially in the field of Byzantine machine learning. With a truncation parameter t, the function $\text{TM} : \mathbb{R}^\rho \longrightarrow \mathbb{R}$ averages all but the t smallest and largest values among the ρ initial values where we assume that $f < t < \rho/2$. In [40], the authors derived, assuming *unbiased estimation*, an upper bound on the trimmed mean of ρ scalars with f among them being possibly Byzantine, by leveraging results on order statistics. We extend their result to *biased estimation*. Recall that $\mathcal{C}$ denotes the set of correct workers, and for each $i \in \mathcal{C}$ its gradient $G_i(w)$ at

parameter $w \in \mathbb{R}^d$ is defined by (2). We also denote the gradients sent by a Byzantine worker i by $G_i(w)$, which however may be arbitrary.

Proposition. *Let $\rho > 2f$, then for all $w \in \mathbb{R}^d$ and $j \in [d]$ and for any $\mathcal{Q} \subset [n]$, such that $|\mathcal{Q}| > \rho$ we have*

$$\mathbb{E}\left[\left(\mathrm{TM}_j - \{\nabla \mathcal{L}(w)\}_j\right)^2\right] \leq \Delta(\rho, f)\left(\frac{1}{|\mathcal{C}\cap\mathcal{Q}|}\sum_{i\in\mathcal{C}\cap\mathcal{Q}} \mathbb{E}\left[|\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j|^2\right]\right)$$

where $\Delta(\rho, f) = \frac{2|\mathcal{C}\cap\mathcal{Q}|(t+|\mathcal{C}\cap\mathcal{Q}|)}{(|\mathcal{C}\cap\mathcal{Q}|-t)^2}$ and TM_j denotes TM: $\left(\{G_i(w)\}_j ; i \in \mathcal{Q}\right)$.

Proof. Let us consider $w \in \mathbb{R}^d$, $j \in [d]$ and $\mathcal{Q} \subset [n]$, such that $|\mathcal{Q}| > \rho$. From a prior result [40, proof of Theorem 1], we have

$$\begin{aligned}&\left(\mathrm{TM}_j - \{\nabla \mathcal{L}(w)\}_j\right)^2\\ &\leq \frac{2}{(|\mathcal{C}\cap\mathcal{Q}| - t)^2}\left(\left(\sum_{i\in\mathcal{C}\cap\mathcal{Q}}(\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j)\right)^2 + t\sum_{i\in\mathcal{C}\cap\mathcal{Q}}(\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j)^2\right). \end{aligned} \tag{5}$$

Furthermore, from Jensen's inequality and the fact that correct workers sample mini-batches independently, we get

$$\mathbb{E}\left[\left(\sum_{i\in\mathcal{C}\cap\mathcal{Q}}(\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j)\right)^2\right] \leq |\mathcal{C}\cap\mathcal{Q}|\sum_{i\in\mathcal{C}\cap\mathcal{Q}}\mathbb{E}\left[|\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j|^2\right]. \tag{6}$$

Hence taking the expectation on both sides in (5) and using (6) we get the expected result. □

B.2 Proof of Lemma 1

We first define the expectations used in what follows:

- $\mathbb{E}_k[\cdot]$: the conditional expectation given the history $\mathcal{P}_k = \left\{w_1, \ldots, w_k; \mathrm{AG}_{w_1}, \ldots, \mathrm{AG}_{w_{k-1}}; \mathcal{B}_k\right\}$
- $\mathbb{E}_{\mathcal{B}_k}$: the conditional expectation given $\mathcal{P}_k \setminus \mathcal{B}_k$
- $\mathbb{E}[\cdot] = \mathbb{E}_{\mathcal{B}_1}\mathbb{E}_1[\ldots\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k\ldots]$: the expectation over the total randomness of the algorithm

To prove Lemma 1 we first show the following intermediate result.

Lemma 2. *Suppose that Assumptions 3 and 4 hold true. Consider the k-th step of Algorithm 1 and $\mathcal{B}_k$ the block computed by the server at step k. If AR is $\Delta(v, f)$ robust at w_k then*

$$\mathbb{E}_k\left[\left\|\mathrm{AG}_{w_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] \leq \Delta(v, f)\left(U_{w_k} + V_{w_k}\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right) \tag{7}$$

where $s_i' := \frac{s_i(|\mathcal{S}_i|-1)}{|\mathcal{S}_i|-s_i}$, $U_{w_k} = \frac{1}{|\mathcal{C}|}\sum_{j\in[d]}\left(\sum_{i\in\mathcal{C}}\left(\frac{M_{w_k}}{s_i'} + \delta_{i,j}^2\right)\right)$, $V_{w_k} = \frac{1}{|\mathcal{C}|}\sqrt{\sum_{j\in[d]}\left(\sum_{i\in\mathcal{C}}\frac{Q_{w_k}}{s_i'}\right)^2}$, and $\mathcal{C}$ denotes the set of honest workers.

Proof. First, we consider an arbitrary correct worker i and block $\mathcal{B}_k \subseteq [d]$. Recall that $G_i(w_k) = \frac{1}{s_i}\sum_{x\in\zeta_i}\nabla\ell(w_k, x)$ where ζ_i is a mini-batch of s_i data points sampled randomly from $\mathcal{S}_i$. Note that

$$\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] = \underbrace{\mathbb{E}_{\zeta_i}\left[\left\|[G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right]\right\|^2\right]}_{\mathbf{A}} + \underbrace{\left\|\mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2}_{\mathbf{B}}. \tag{8}$$

We obtain below upper bounds on **A** and **B**.
A). Note that ζ_i is a mini-batch obtained by sampling s_i random data points from $\mathcal{S}_i$ without replacements, where $s_i = |\zeta_i|$. Furthermore, by definition of $[\cdot]_{\mathcal{B}_k}$, $\forall y \in \mathbb{R}^d$ we have $\|[y]_{\mathcal{B}_k}\|^2 = \sum_{j\in\mathcal{B}_k} (\{y\}_j)^2$. Hence, from [**?**], we have

$$\mathbb{E}_{\zeta_i}\left[\|[G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right]\|^2\right] \leq \mathbb{E}_{z\sim\mathcal{S}_i}\left[\sum_{j\in\mathcal{B}_k}\left(\{\nabla l(w_k,\, z)\}_j - \mathbb{E}_{x\sim\mathcal{S}_i}\left[\{\nabla l(w_k, x)\}_j\right]\right)^2\right]\left(\frac{|\mathcal{S}_i| - s_i}{s_i(|\mathcal{S}_i|-1)}\right).$$

Recall that for all $s_i' := \frac{s_i(|\mathcal{S}_i|-1)}{|\mathcal{S}_i|-s_i}$. Finally, substituting from Assumption 4 we obtain that

$$\mathbb{E}_{\zeta_i}\left[\|[G_i(w_k)]_{\mathcal{B}_k} - \mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right]\|^2\right] \leq \sum_{j\in\mathcal{B}_k}\left(\frac{M_{w_k}}{s_i'} + \frac{Q_{w_k}}{s_i'}\left(\{\nabla\mathcal{L}(w_k)\}_j\right)^2\right) = \frac{b_k M_{w_k}}{s_i'} + \frac{Q_{w_k}}{s_i'}\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \tag{9}$$

where b_k is the size of block $\mathcal{B}_k$.
B). Similarly, we note that

$$\|\mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right] - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 = \|\mathbb{E}_{x\sim\mathcal{S}_i}\left[[\nabla\ell(w_k,\, x)]_{\mathcal{B}_k}\right] - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2$$

Substituting from Assumption 3 above, we obtain that

$$\|\mathbb{E}_{\zeta_i}\left[[G_i(w_k)]_{\mathcal{B}_k}\right] - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \leq \sum_{j\in\mathcal{B}_k}\delta_{i,j}^2. \tag{10}$$

A+B). Substituting from (9) and (10) in (8) we obtain that

$$\mathbb{E}_{\zeta_i}\left[\|[G_i(w_k)]_{\mathcal{B}_k} - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2\right] \leq \frac{b_k M_{w_k}}{s_i'} + \sum_{j\in\mathcal{B}_k}\delta_{i,j}^2 + \frac{Q_{w_k}}{s_i'}\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2. \tag{11}$$

As $\mathcal{B}_k$ is an arbitrary subset of $[d]$, (11) implies that for all $j \in \mathcal{B}_k$,

$$\mathbb{E}_{\zeta_i}\left[(\{G_i(w_k)\}_j - \{\nabla\mathcal{L}(w_k)\}_j)^2\right] \leq \left(\frac{M_{w_k}}{s_i'} + \delta_{i,j}^2\right) + \frac{Q_{w_k}}{s_i'}\left(\{\nabla\mathcal{L}(w_k)\}_j\right)^2. \tag{12}$$

Next, recall that in the algorithm for each $j \in \mathcal{B}_k$ the server receives the partial derivatives from at least v workers denoted by set $\mathcal{Q}_j$, i.e., $|\mathcal{Q}_j| \geq v$. We denote the set of correct workers that send the j-th coordinate of their gradients by $\mathcal{C}_j = \mathcal{C} \cap \mathcal{Q}_j$. Since (11) holds for an arbitrary correct worker $i \in \mathcal{C}$, we obtain that

$$\frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\mathbb{E}_{\zeta_i}\left[(\{G_i(w_k)\}_j - \{\nabla\mathcal{L}(w_k)\}_j)^2\right] \leq \frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_{w_k}}{s_i'} + \delta_{i,j}^2\right) + \frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\{\nabla\mathcal{L}(w_k)\}_j^2. \tag{13}$$

Now, as AR is assumed $\Delta(v,f)$-robust at w_k, by Definition 1,

$$\mathbb{E}_k\left[(\{\mathrm{AG}_{w_k}\}_j - \{\nabla\mathcal{L}(w_k)\}_j)^2\right] \leq \Delta(v,f)\left(\frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\mathbb{E}_{\zeta_i}\left[(\{G_i(w_k)\}_j - \{\nabla\mathcal{L}(w_k)\}_j)^2\right]\right).$$

Substituting from (13) above we obtain that

$$\mathbb{E}_k(\{\mathrm{AG}_{w_k}\}_j - \{\nabla\mathcal{L}(w_k)\}_j)^2 \leq \Delta(v,f)\left(\frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_{w_k}}{s_i'} + \delta_{i,j}^2\right) + \frac{1}{|\,\mathcal{C}_j\,|}\sum_{i\in\mathcal{C}_j}\left(\frac{Q_{w_k}}{s_i'}\{\nabla\mathcal{L}(w_k)\}_j^2\right)\right). \tag{14}$$

By summing both sides in (14) over all $j \in \mathcal{B}$ we obtain that

$$\mathbb{E}_k[\|[\mathrm{AG}_{w_k}]_{\mathcal{B}_k} - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] \leq$$
$$\Delta(v,f)\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_{w_k}}{s_i'}+\delta_{i,j}^2\right)\right)+\sum_{j\in\mathcal{B}_k}\left(\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\right)\{\nabla\mathcal{L}(w_k)\}_j^2\right)\right). \tag{15}$$

Using Cauchy-Schwartz $|\sum_i a_i b_i| \leq \left(\sum_i a_i^2\right)^{\frac{1}{2}}\left(\sum_i b_i^2\right)^{\frac{1}{2}}$, we have

$$\sum_{j\in\mathcal{B}_k}\left(\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\right)\{\nabla\mathcal{L}(w_k)\}_j^2\right) \leq \left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\right)^2\right)^{\frac{1}{2}}\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w_k)\}_j^4\right)^{\frac{1}{2}}. \tag{16}$$

Recall that $\left(\sum_i a_i^2\right)^{\frac{1}{2}} \leq \sum_i |a_i|$. Thus,

$$\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w_k)\}_j^4\right)^{\frac{1}{2}} \leq \sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w_k)\}_j^2 \tag{17}$$

Recall that $[\mathrm{AG}_{w_k}]_{\mathcal{B}_k} = \mathrm{AG}_{w_k}$. Hence, substituting from (16) and (17) in (15), we have

$$\mathbb{E}_k[\|\mathrm{AG}_{w_k} - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2] \leq$$
$$\Delta(v,f)\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\left(\frac{M_{w_k}}{s_i'}+\delta_{i,j}^2\right)\right)+\left(\sum_{j\in\mathcal{B}_k}\left(\frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_j}\frac{Q_{w_k}}{s_i'}\right)^2\right)^{\frac{1}{2}}\left(\sum_{j\in\mathcal{B}_k}\{\nabla\mathcal{L}(w_k)\}_j^2\right)\right)$$
$$\leq \Delta(v,f)\left(\frac{1}{|\mathcal{C}|}\sum_{j\in[d]}\left(\sum_{i\in\mathcal{C}}\left(\frac{M_{w_k}}{s_i'}+\delta_{i,j}^2\right)\right)+\frac{1}{|\mathcal{C}|}\left(\sum_{j\in[d]}\left(\sum_{i\in\mathcal{C}}\frac{Q_{w_k}}{s_i'}\right)^2\right)^{\frac{1}{2}}\left(\sum_{j\in[d]}\{\nabla\mathcal{L}(w_k)\}_j^2\right)\right)$$

which concludes the proof.

□

We are now ready to prove Lemma 1, which is stated again below for readability.

Lemma. *Suppose that Assumptions 3 and 4 hold true. Consider the k-th step of Algorithm 1 and $\mathcal{B}_k$ the block computed by the server at step k. If* AR *is $\Delta(v,f)$-robust at w_k and $w_k \in \mathcal{W}$ then*

$$\mathbb{E}_k[\langle \mathrm{AG}_{w_k}, [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle] > \frac{1}{2}\left((1-\Delta(v,f)V_{w_k})\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \Delta(v,f)U_{w_k}\right).$$

Proof. From Lemma 2 we have

$$\underbrace{\Delta(v,f)\left(U_w + V_w\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\|^2\right)}_{:=r^2} \geq \mathbb{E}_k[\|\mathrm{AG}_w - [\nabla\mathcal{L}(w)]_{\mathcal{B}}\|^2] \geq \|\mathbb{E}_k[\mathrm{AG}_w] - [\nabla\mathcal{L}(w)]_{\mathcal{B}}\|^2 \tag{18}$$

where the second inequality follows from Jensen's inequality. The fact that $\|\mathbb{E}_k[\mathrm{AG}_w] - [\nabla\mathcal{L}(w)]_{\mathcal{B}}\|^2 \leq r^2$ means that $\mathbb{E}_k[\mathrm{AG}_w]$ belongs to a ball centered at $[\nabla\mathcal{L}(w)]_{\mathcal{B}}$ with radius r, illustrated in Figure 9 below.

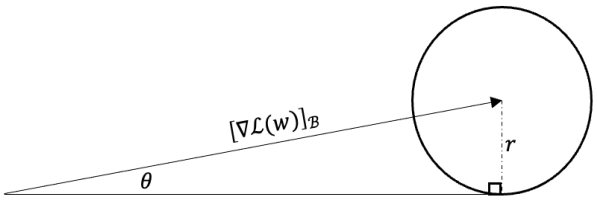


Figure 9: When $\|\mathbb{E}_k[\mathrm{AG}_{w_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| \leq r$, $\mathbb{E}_k[\mathrm{AG}_{w_k}]$ belongs to a ball centered at $[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}$ with radius r.

From (18) we obtain that

$$\|\mathbb{E}_k[\mathrm{AG}_{w_k}] - [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 = \|\mathbb{E}_k[\mathrm{AG}_{w_k}]\|^2 - 2\langle \mathbb{E}[\mathrm{AG}_{w_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \leq r^2$$

Thus,

$$2\,\langle \mathbb{E}_k[\mathrm{AG}_{w_k}],\, [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle \geq \|\mathbb{E}_k[\mathrm{AG}_{w_k}]\|^2 + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r^2. \tag{19}$$

As we assume that $w_k \in \mathcal{W}, \quad \Delta(v,f)U_{w_k} \leq (1 - \Delta(v,f)V_{w_k})\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2$. Thus,

$$\frac{r}{\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|} \leq 1. \tag{20}$$

Owing to (20), we consider $\theta \in [0,\, \pi/2]$ such that $\sin\theta = \frac{r}{\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|}$. Furthermore, from the triangle inequality, $\|\mathbb{E}_k[\mathrm{AG}_{w_k}]\| \geq \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| - r$. Therefore, $\|\mathbb{E}_k[\mathrm{AG}_{w_k}]\| \geq (1-\sin\theta)\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| \geq 0$. Using these in (19) we obtain that

$$\begin{aligned}
2\langle \mathbb{E}_k[\mathrm{AG}_{w_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle &\geq \|\mathbb{E}_k[\mathrm{AG}_{w_k}]\|^2 + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r^2 \\
&\geq (1-\sin\theta)^2\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 + \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \sin^2\theta\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 \\
&= 2(1-\sin\theta)\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 .
\end{aligned}$$

Thus,

$$\langle \mathbb{E}_k[\mathrm{AG}_{w_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle \geq (1-\sin\theta)\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 .$$

Substituting $\sin\theta = \frac{r}{\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|}$ above we obtain that

$$\langle \mathbb{E}_k[\mathrm{AG}_{w_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle \geq \|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\| \geq \frac{1}{2}\left(\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - r^2\right).$$

Recall that $r^2 := \Delta(v,f)\left(U_{w_k} + V_{w_k}\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2\right)$. Substituting this above concludes the proof, i.e., we have

$$\langle \mathbb{E}_k[\mathrm{AG}_{w_k}], [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\rangle \geq \frac{1}{2}\left((1 - V_{w_k}\Delta(v,f))\,\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\|^2 - \Delta(v,f)U_{w_k}\right).$$

□

B.3 Proof of Theorem 1

Theorem. *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate γ. If $\gamma < \frac{1}{dL}$ and $\Delta(v,f)V < 1$ then*

$$\mathbb{E}\left[\mathcal{L}(w_T) - \mathcal{L}^*\right] \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)^{T-1}\left(\mathcal{L}(w_1) - \mathcal{L}^* - H\right) + H$$

where $\mathcal{L}^*$ denotes the minimum value of $\mathcal{L}$ on $\mathbb{R}^d$, $b = \min\limits_{k\in[T]}\left[|\mathcal{B}_k|\right], U = \max\limits_{k\in[T]} U_{w_k}, V = \max\limits_{k\in[T]} V_{w_k}$, and $H = \frac{d\Delta(v,f)U}{2b\mu(1-\Delta(v,f)V)}$.

Proof. Recall the definition of set $\mathcal{W}$ from (4), i.e.,

$$\mathcal{W} := \left\{w \in \mathbb{R}^d;\ \min_{\mathcal{B}\subseteq[d]}\left\{(1-\Delta(v,f)V_w)\left\|[\nabla\mathcal{L}(w)]_{\mathcal{B}}\right\|^2 - \Delta(v,f)U_w\right\} > 0\right\}.$$

To prove the theorem, we separately consider below the two cases: **A)** when $w_k \in \mathcal{W}$ for all $k \in [T]$, and **B)** there exists $s \in [T]$ such that $w_s \in \mathcal{W}$.

A). All the elements of the sequence $\{w_k\}_{k\in[T]}$ are in the subspace $\mathcal{W}$. For all $k \in [T-1]$, we have $w_{k+1} = w_k - \gamma \mathrm{AG}_{w_k}$. Now, we consider an arbitrary $k \in [T-1]$. Under Assumption 1, i.e., component-wise Lipschitz continuity of $\nabla\mathcal{L}(w)$ with coefficient L, and the fact that $\mathrm{AG}_{w_k} = [\mathrm{AG}_{w_k}]_{\mathcal{B}_k}$ we have [30, Section 2.1]

$$\mathcal{L}(w_{k+1}) = \mathcal{L}(w_k - \gamma\mathrm{AG}_{w_k}) \leq \mathcal{L}(w_k) - \gamma\langle[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathrm{AG}_{w_k}\rangle + \gamma^2\frac{L_{\mathcal{B}_k}}{2}\left\|\mathrm{AG}_{w_k}\right\|^2 \tag{21}$$

where $L_{\mathcal{B}_k}$ is the Lipschitz coefficient of the truncated gradient $[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}$. Using the fact that $\left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k} - \mathrm{AG}_{w_k}\right\|^2 = \left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - 2\langle[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathrm{AG}_{w_k}\rangle + \left\|\mathrm{AG}_{w_k}\right\|^2$, (21) becomes

$$\begin{aligned}\mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) - \gamma(1-\gamma L_{\mathcal{B}_k})\langle[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathrm{AG}_{w_k}\rangle \\ +\gamma^2\frac{L_{\mathcal{B}_k}}{2}\left(\left\|\mathrm{AG}_{w_k} - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - \left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right).\end{aligned} \tag{22}$$

By taking the conditional expectation $\mathbb{E}_k[\cdot]$ on both sides in (22) we obtain

$$\begin{aligned}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \gamma(1-\gamma L_{\mathcal{B}_k})\langle[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}, \mathbb{E}_k[\mathrm{AG}_{w_k}]\rangle \\ +\gamma^2\frac{L_{\mathcal{B}_k}}{2}\left(\mathbb{E}_k\left[\left\|\mathrm{AG}_{w_k} - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] - \left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right)\end{aligned} \tag{23}$$

Recall that $L_{\mathcal{B}_k} \leq \sqrt{|\mathcal{B}_k|}L \leq \sqrt{d}L$ where $L = \max\limits_{i\in[d]} L_i$. Then, using Lemma 1 and the fact that $\gamma < \frac{1}{\sqrt{d}L}$ we obtain:

$$\begin{aligned}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{1}{2}\gamma(1-\gamma\sqrt{d}L)((1-\Delta(v,f)V_{w_k})\left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - \Delta(v,f)U_{w_k}) \\ +\gamma^2\frac{\sqrt{d}L}{2}\left(\mathbb{E}_k\left[\left\|\mathrm{AG}_{w_k} - [\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] - \left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right)\end{aligned} \tag{24}$$

Using Lemma 2 in (24), we also have:

$$\begin{aligned}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{1}{2}\gamma(1-\gamma\sqrt{d}L)\left((1-\Delta(v,f)V_{w_k})\left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 - \Delta(v,f)U_{w_k}\right) \\ +\gamma^2\frac{\sqrt{d}L}{2}\left(\Delta(v,f)\left(U_{w_k} + V_{w_k}\left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right) - \left\|[\nabla\mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right)\end{aligned} \tag{25}$$

Rearranging the terms gives:

$$\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\gamma}{2}(1-\Delta(v,f)V_{w_k}) \left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2 + \frac{\gamma\Delta(v,f)U_{w_k}}{2} \tag{26}$$

Now, introducing $\mathbb{E}_{\mathcal{B}_k}$ the conditional expectation given $\mathcal{P}_k \setminus \mathcal{B}_k$ we get

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k\left[\mathcal{L}(w_{k+1})\right] \leq \mathcal{L}(w_k) - \frac{\gamma}{2}(1-\Delta(v,f)V_{w_k})\,\mathbb{E}_{\mathcal{B}_k}\left[\left\|[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}\right\|^2\right] + \frac{\gamma\Delta(v,f)U_{w_k}}{2} \tag{27}$$

$$\leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k}\left[|B_k|\right]}{d}\frac{\gamma}{2}(1-\Delta(v,f)V_{w_k}) \left\|\nabla \mathcal{L}(w_k)\right\|^2 + \frac{\gamma\Delta(v,f)U_{w_k}}{2} \tag{28}$$

Using the Polyak-Lojasiewicz inequality in (28) gives:

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right]}{d}\mu\gamma(1-\Delta(v,f)V_{w_k})(\mathcal{L}(w_k)-\mathcal{L}^*) + \frac{\gamma\Delta(v,f)U_{w_k}}{2} \tag{29}$$

Let $b = \min\limits_{k\in[T]} \mathbb{E}_{\mathcal{B}_k}\left[|B_k|\right], U = \max\limits_{k\in[T]} U_{w_k}$ and $V = \max\limits_{k\in[T]} V_{w_k}$. Substituting from the above, we get

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)(\mathcal{L}(w_k)-\mathcal{L}^*) + \frac{\gamma\Delta(v,f)U}{2} \tag{30}$$

If we now subtract $H = \frac{d\Delta(v,f)U}{2b\mu(1-\Delta(v,f)V}$ and $\mathcal{L}^*$ from both sides of (30), we obtain:

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)(\mathcal{L}(w_k) - \mathcal{L}^* - H) \tag{31}$$

As $\mathbb{E}\left[\cdot\right] = \mathbb{E}_{\mathcal{B}_1}\mathbb{E}_1\left[\ldots\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k\ldots\right]$, from above we obtain that

$$\mathbb{E}[\mathcal{L}(w_{k+1})] - \mathcal{L}^* - H \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)(\mathbb{E}[\mathcal{L}(w_k)] - \mathcal{L}^* - H) \tag{32}$$

By assumption, $\Delta(v,f)V < 1$. We also know that $\mu < \min_{k\in[T]} L_{\mathcal{B}_k} \leq \sqrt{d}L$, which means that: $0 < \frac{b}{d}\mu\gamma(1-\Delta(v,f)V) < 1$. Inequality (32) is therefore a contraction inequality. By applying it repeatedly through iterations $k \in [T-1]$, we obtain:

$$\mathbb{E}\left[\mathcal{L}(w_T)\right] - \mathcal{L}^* \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)^{T-1}(\mathcal{L}(w_1) - \mathcal{L}^* - H) + H. \tag{33}$$

B). There exists at least one element of the sequence $\{w_k\}_{k\in[T]}$ that belongs to the subspace $\mathbb{R}^d \setminus \mathcal{W}$. Let us denote by s the biggest integer such that $w_s \in \mathbb{R}^d \setminus \mathcal{W}$, then by definition of $\mathcal{W}$, for all $\mathcal{B} \subset [d]$

$$\left\|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}}\right\|^2 \leq \frac{\Delta(v,f)U_{w_s}}{1-\Delta(v,f)V_{w_s}}. \tag{34}$$

Thus,

$$\mathbb{E}_{\mathcal{B}_s}\left[\left\|[\nabla \mathcal{L}(w_s)]_{\mathcal{B}_s}\right\|^2\right] \leq \frac{\Delta(v,f)U_{w_s}}{1-\Delta(v,f)V_{w_s}} \tag{35}$$

$$\frac{\mathbb{E}_{\mathcal{B}_s}\left[b_s\right]}{d}\left\|\nabla \mathcal{L}(w_s)\right\|^2 \leq \frac{\Delta(v,f)U_{w_s}}{1-\Delta(v,f)V_{w_s}} \tag{36}$$

$$\left\|\nabla \mathcal{L}(w_s)\right\|^2 \leq \frac{d\Delta(v,f)U_{w_s}}{\mathbb{E}_{\mathcal{B}_s}\left[|\mathcal{B}_s|\right](1-\Delta(v,f)V_{w_s})} \leq \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}. \tag{37}$$

Using the Polyak-Lojasiewicz inequality, we have

$$\left\|\nabla \mathcal{L}(w_s)\right\|^2 \geq 2\mu(\mathcal{L}(w_s) - \mathcal{L}^*). \tag{38}$$

Combining (37) and (38) gives

$$\mathcal{L}(w_s) - \mathcal{L}^* \leq \underbrace{\frac{d\Delta(v,f)U}{2b\mu(1-\Delta(v,f)V)}}_{H}. \tag{39}$$

If $s = T$, then the above implies

$$\mathcal{L}(w_T) - \mathcal{L}^* \leq \underbrace{\frac{d\Delta(v,f)U}{2b\mu(1-\Delta(v,f)V)}}_{H}. \tag{40}$$

Otherwise, recall that for all $k \in [T]$ where $k > s$ we have $w_k \in \mathcal{W}$. Thus by applying the same reasoning as in **A)** we get

$$\mathbb{E}\left[\mathcal{L}(w_T)\right] - \mathcal{L}^* \leq \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)^{T-s} (\mathbb{E}[\mathcal{L}(w_s)] - \mathcal{L}^* - H) + H. \tag{41}$$

Finally by substituting (39) in (41) we get $\mathbb{E}\left[\mathcal{L}(w_T)\right] - \mathcal{L}^* \leq H$.

A+B).

From (33), (40) and (41) we obtain, $\forall w \in \mathbb{R}^d$:

$$\mathbb{E}\left[\mathcal{L}(w_T)\right] - \mathcal{L}^* \leq \max\left\{\left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H, H\right\} \tag{42}$$

$$= \left(1 - \frac{b}{d}\mu\gamma(1-\Delta(v,f)V)\right)^{T-1} (\mathcal{L}(w_1) - \mathcal{L}^* - H) + H \tag{43}$$

which concludes the proof. □

B.4 Proof of Theorem 2

Theorem (Non-convex convergence)**.** *Suppose that Assumptions 1, 3 and 4 hold. If* $\gamma < \frac{1}{\sqrt{d}L}$ *and* $\Delta(v,f)V < 1$ *then*

$$\min_{k\in[T]} \mathbb{E}\left[\left\|\nabla \mathcal{L}(w_k)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Tb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}$$

where $U = \max_{k\in[T]} U_{w_k}$, $V = \max_{k\in[T]} V_{w_k}$ *and* $b = \min_{k\in[T]} \mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right]$.

Proof. As in the proof of the previous theorem we decompose this proof into two parts:
A). All the elements of the sequence $\{w_k\}_{k\in[T]}$ are in the subspace $\mathcal{W}$.

Let us recall that from (28), for any $k \in [T]$ we have

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{\mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right]}{2d}\gamma(1-\Delta(v,f)V_{w_k})\left\|\nabla \mathcal{L}(w_k)\right\|^2 + \frac{\gamma\Delta(v,f)U_{w_k}}{2}. \tag{44}$$

Let $U = \max_{k\in[T]} U_{w_k}$, $V = \max_{k\in[T]} V_{w_k}$, and $b = \min_{k\in[T]} \mathbb{E}_{\mathcal{B}_k}\left[|\mathcal{B}_k|\right]$. Using these upper bounds, for any $k \in [T]$ we get

$$\mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})] \leq \mathcal{L}(w_k) - \frac{b\gamma}{2d}(1-\Delta(v,f)V)\left\|\nabla \mathcal{L}(w_k)\right\|^2 + \frac{\gamma\Delta(v,f)U}{2}. \tag{45}$$

Rearranging the terms gives

$$\left\|\nabla \mathcal{L}(w_k)\right\|^2 \leq \frac{2d(\mathcal{L}(w_k) - \mathbb{E}_{\mathcal{B}_k}\mathbb{E}_k[\mathcal{L}(w_{k+1})])}{b\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{46}$$

As $\mathbb{E}[\cdot] = \mathbb{E}_1[\dots \mathbb{E}_t[\cdot]\dots]$, from above we obtain that

$$\mathbb{E}\left\|\nabla \mathcal{L}(w_k)\right\|^2 \leq \frac{2d(\mathbb{E}[\mathcal{L}(w_k)] - \mathbb{E}[\mathcal{L}(w_{k+1})])}{b\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{47}$$

As the above is true for all $k \in [T]$, we can sum both sides through $[1, \cdots, T]$ to get

$$\sum_{k=1}^{T} \mathbb{E}\left\|\nabla \mathcal{L}(w_k)\right\|^2 \leq \frac{2d(\mathbb{E}[\mathcal{L}(w_1)] - \mathbb{E}[\mathcal{L}(w_{T+1})])}{b\gamma(1-\Delta(v,f)V)} + T\frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{48}$$

Dividing the last inequality by T gives:

$$\mathbb{E}\left[\frac{1}{T}\sum_{k=1}^{T}\left\|\nabla \mathcal{L}(w_k)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathbb{E}[\mathcal{L}(w_{T+1})])}{Tb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{49}$$

And using the fact that $\mathbb{E}[\mathcal{L}(w_{T+1})] > \mathcal{L}(w_*)$:

$$\mathbb{E}\left[\frac{1}{T}\sum_{k=1}^{T}\left\|\nabla \mathcal{L}(w_k)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{Tb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}, \tag{50}$$

which also means that

$$\min_{k\in[T]} \mathbb{E}\left[\left\|\nabla \mathcal{L}(w_k)\right\|^2\right] \leq \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}. \tag{51}$$

B). There exists at least one element w_s of the sequence $\{w_k\}_{k\in[T]}$ in the subspace $\mathbb{R}^d \setminus \mathcal{W}$.
By construction of $\mathcal{W}$, as in (37), we have:

$$\left\|\nabla \mathcal{L}(w_s)\right\|^2 < \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{52}$$

A+B). By combining (50) and (52), we obtain, $\forall w \in \mathbb{R}^d$:
which also means that:

$$\min_{1<i<T} \mathbb{E}\left[\left\|\nabla \mathcal{L}(w_i)\right\|^2\right] \leq \max\left\{\frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}, \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)}\right\} \tag{53}$$

$$= \frac{2d(\mathcal{L}(w_1) - \mathcal{L}(w_*))}{kb\gamma(1-\Delta(v,f)V)} + \frac{d\Delta(v,f)U}{b(1-\Delta(v,f)V)} \tag{54}$$

and that concludes the proof. □

C Additional details on experiments

In this work, we choose linear regression, logistic regression, support vector machine and neural networks as ML models to test the performance of several instances of HGO against SGD. We also use the datasets CIFAR-10, MNIST, BOSTON and PHISHING. In the following, we describe their properties as well as the pre-processing steps adopted for each one of them.

C.1 ML models

Linear regression. This linear model is used to predict a scalar value based on a set of explanatory variables. It is very efficient when the relation between the input features and the output is linear. The cost function in this case is a simple squared euclidean norm of the residuals:

$$C(w) = \|Xw - y\|^2$$

Logistic regression. This model computes the probabilities for classification problems with two possible outcomes. Instead of using a simple linear hypothesis Xw, as in linear regression, a sigmoid function is applied to this linear hypothesis: $h_w(X) = \frac{1}{1+e^{-Xw}}$. Combined with the logistic loss, the cost function of this model can be written as follows:

$$C(w) = \frac{1}{m} \sum_{i=1}^{m} -y_i \log(h_w(X_i)) + (1 - y_i) \log(1 - h_w(X_i))$$

Support vector machine. This last model constructs a hyperplane in a high-dimensional space that can be used for classification tasks. In this work, we choose the regularized version, and the cost function is written as follows:

$$C(w) = \frac{1}{2} \|w\|^2 + C \left[\frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(wx_i)) \right]$$

Neural networks. Neural network learns to map a set of inputs to a set of outputs from training dataset. A deep neural network consists of an input layer representing the set of input features X, an output layer representing the number of classification classes, and a number of hidden layers with multiple hidden units, as depicted in Figure 10. Each unit computes its value based on the combination of values from previous layers and an activation function. The most common activation functions are:

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{(-z)}}$
- Tanh: $\tanh z = \frac{e^{(z)} - e^{(-z)}}{e^{(z)} + e^{(-z)}}$
- ReLU (Rectified Linear Unit): $ReLU(z) = \max(0, z)$

C.2 Datasets and pre-processing

To conduct an extensive and diversified set of experiments, we have considered multiple datasets. Each dataset is used with a specific ML model.

MNIST dataset. The MNIST dataset consists in 10 categories of digits, from 0 to 9, represented by a set of 784 features, with a total of 70,000 data samples (60,000 for training and 10,000 for testing). The training set is shuffled and normalized; then, each worker is assigned a non-overlapping i.i.d. sample from the training set, drawn uniformly. MNIST is used with logistic regression to perform a binary classification, and neural networks for a full classification.

BOSTON dataset. The BOSTON dataset contains information (collected by the US Census Service) about housing in the area of Boston (Massachusetts). The dataset contains 506 data samples and 14 feature variables. After standardizing its features, each worker is assigned (randomly) a non-overlapping i.i.d. sample from the training set. The dataset is used in conjunction with a linear regression model to predict the price of houses.

PHISHING dataset. The PHISHING dataset contains fraudulent attempts to obtain sensitive information (e.g., usernames, passwords, and credit card details) through email spoofing, instant messaging and text messaging. The dataset contains 8844 fraudulent attempts, characterized by 68 features. All features have been scaled by their maximum absolute value and normalized. Then, each worker is assigned (randomly) a non-overlapping i.i.d. sample from the training set. The dataset is used with a support vector machine (SVM) model to check whether the URL is phishing or not.

CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The training set is shuffled and normalized; then, each worker is assigned a non-overlapping i.i.d. sample from the training set, drawn uniformly.

In the experiments involving non identically distributed datasets, each worker will only be receiving a random sample of the training set, containing only a few classes.

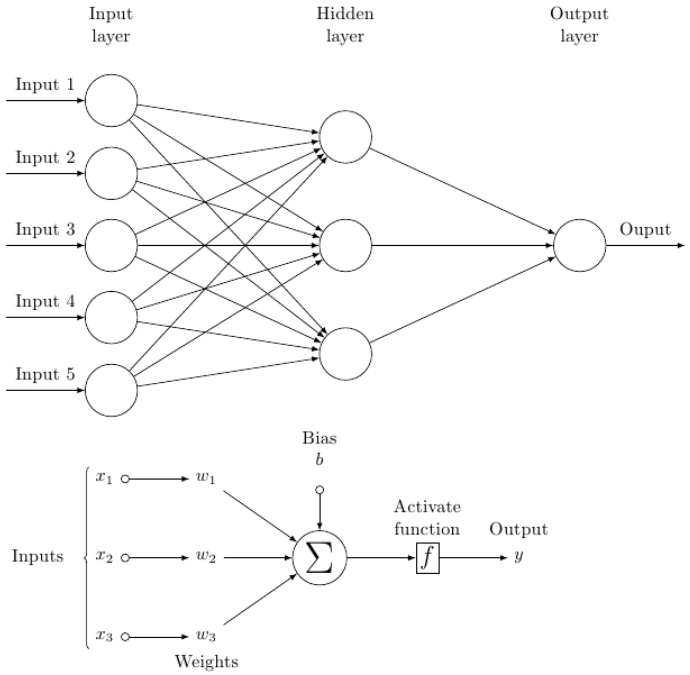


Figure 10: A feed-forward neural network with a 5-inputs layer, one hidden layer with 3 neurons and one output layer with only one class. Each neuron is composed of a transfer function summing all the inputs and a bias, followed by an activation function.

C.3 Specification of experiments

Block generation for neural networks. In the case of neural networks, block generation is different. As a matter of fact, stacking all the weights into a single vector and picking randomly a block constructed as defined above will result in an unbalanced updates between the network layers. Instead, we use a vector containing the number of coordinates (weights) to be updated in each layer. For example, in a feed-forward neural network with 784 inputs, one hidden layer with 30 neurons and an output with 10 classes (a total of 3 layers), a block size will be written in the following format: [98,20,10]. This means that 98 out of 784 weights in the first layer, 20 out of 30 weights in the second layer and all the weights in the last layer are selected randomly and updated at each round.

Runs. Because of the stochastic nature of the algorithms, each experiment is run 10 times. We compute the mean and the standard deviation of the metric used in each experiment, and we plot the mean as well as a confidence interval.

Metrics. For the linear regression model, the evaluation metric is the cost function. For the classification tasks using either logistic regression, support vector machine or neural networks, the evaluation metric is accuracy.

C.4 Attacks used in the Byzantine experiments

It is known that Average is not a robust aggregation rule. In fact, as shown in [5], one single Byzantine worker is sufficient to give any value to the aggregated gradient. Moreover, without any computation efforts, a

Byzantine worker can send very large coordinate values. Using this simple strategy can make the ML models diverge. In this work, we used two state-of-the-art attacks that were developed against the most effective robust variants of SGD. In the plots, we always include the convergence graph of HGO with AVERAGE under no attack, as a benchmark to assess the impact of Byzantine workers on the convergence behavior. For the sake of comparison, we also include HGO with the AVERAGE aggregation rule as a reference. In some experiments, this version shows a good performance against those attacks. However, this does not mean that AVERAGE is Byzantine-resilient, and practitioners are aware of it because of proven vulnerabilities.

A little is enough. This attack was presented in [3]. The idea consists in sending Byzantine values that are not too far from the mean, and can be confused with correct values if we consider a normal distribution of correct values. In other words, the Byzantine worker will send values that are far from the mean within an interval of z standard deviations. Algorithm 2 summarizes the attack.

Algorithm 2 "A LITTLE IS ENOUGH" ATTACK

Input: f, n
$n = \lfloor \frac{n}{2} + 1 \rfloor - f$
$z_{max} = \max_z \left(\phi(z) < \frac{n-m-s}{n-m} \right)$
for $j = 1$ to d **do**
  calculate mean (μ_j) and standard deviation (σ_j)
  $[B]_j = \mu_j + z_{max}\sigma_j$
end for
for $i = 1$ to f **do**
  $B_i = B$
end for

Fall of empires. This attack was presented in [41]. This attack tries to manipulate the inner product of the true gradient and the aggregated vector by making it negative. In order to guarantee the progress of any descent algorithm, this inner product must stay positive. The attack is very simple and consists in sending the negative of correct vectors' average, multiplied by a value ϵ. Formally, if $\mathcal{C}$ is the set of correct workers, then:

$$\forall j \in [f], \quad B_j = -\frac{\epsilon}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} G_i$$

C.5 Hardware, libraries and dependencies

All experiments were conducted on a MacBook Pro (2018) with a 2.3 GHz Quad-Core Intel Core i5 processor and 16.0G of RAM. The source code was written in Python (version 3.8) and makes use of usual machine learning libraries, namely: Numpy, Pandas and Scikit-learn.

C.6 Source code

The source code for all experiments is available in the following anonymous GitHub repositories:

- Distributed implementation:
  `https://anonymous.4open.science/r/HgO-5136`
- Android application:
  `https://anonymous.4open.science/r/HgOApp-CC8E`

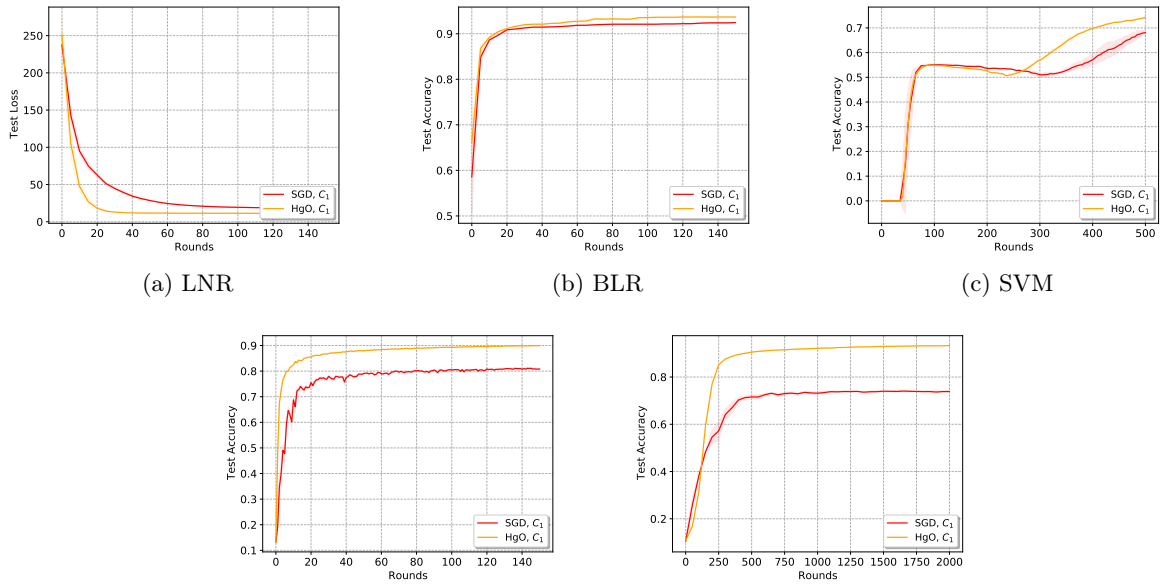
More details are provided in the repositories on how to reproduce the results of the paper.

D Additional set of experiments

In this section, we provide a set of additional experiments comparing all the models and datasets. We use the following combinations in the experiments:

- Linear regression (LNR) on Boston with $n = 12$ workers
- Binary logistic regression (BLR) on Wisconsin-Breast-Cancer with $n = 50$ workers
- Support vector machine (SVM) on Phishing with $n = 12$ workers
- Multinomial logistic regression (MLR) on MNIST with $n = 100$ workers
- Multinomial logistic regression on Fashion-MNIST with $n = 100$ workers
- Multinomial logistic regression on CIFAR-10 with $n = 100$ workers
- Feed-forward neural network (FF-NN) on MNIST with $n = 100$ workers

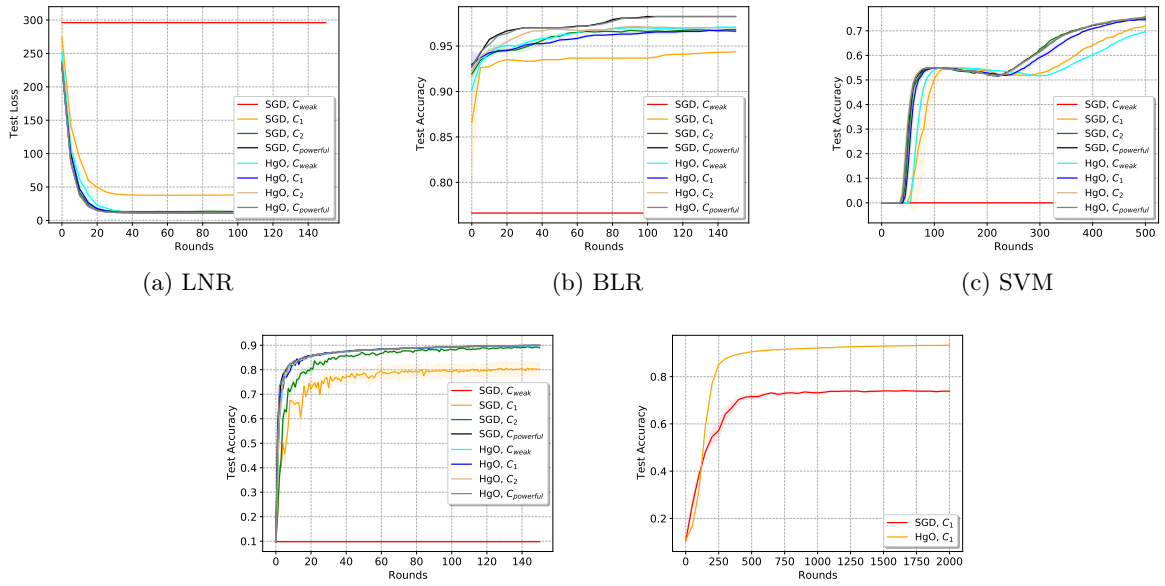
When the dataset is heterogeneously partitioned, we include the abbreviation "HP" in the description.



(a) LNR (b) BLR (c) SVM

(d) MLR(MNIST) - HP (e) FF-NN - HP

Figure 11: Evaluating the performance of HgO against SGD, using the C_1 computation profile.



(a) LNR (b) BLR (c) SVM

(d) MLR(MNIST- HP (e) FF-NN.- HP

Figure 12: Comparing the computation profiles of the network.

E HgO: Android application

In the previous section, we implemented HGO in a simulated distributed environment (on a computer), and evaluated all convergence properties, namely: accuracy, iteration cost, convergence speed and Byzantine resilience. In this section, we report on the deployment of our algorithm as an Android application. As opposed to the simulations, where workers and servers are represented as classes in object-oriented programming, this experiment is a truly distributed training where workers are physically distinct (smartphones) and are connected via WiFi (local network) to a parameter server (PS) hosted in a computer. Since the aggregation at the PS is also lightweight, the PS can be hosted in one of the smartphones involved in the training. This app was developed to test HGO in a network with smartphones, and show that ML is actually possible using these devices. Therefore, the user interface and the app capabilities are very limited, with only essential functionalities to prove our points. A full version of the app is left for future work.

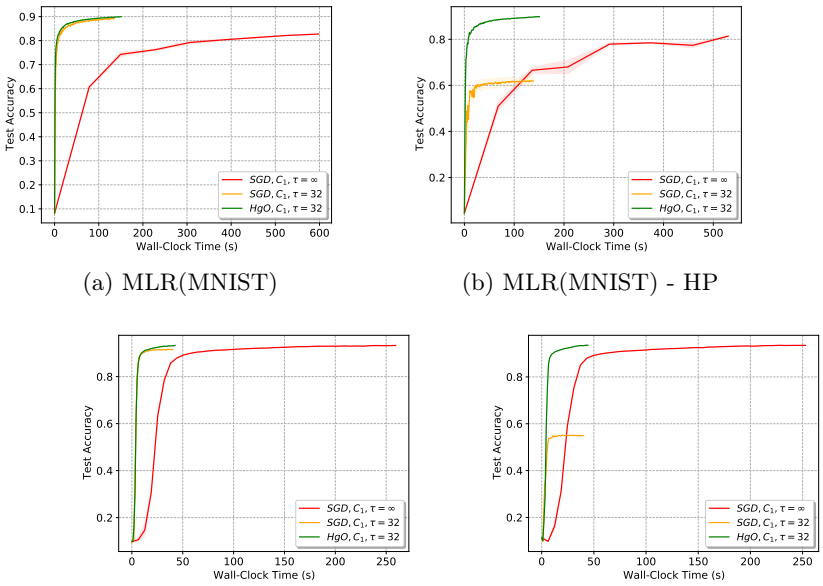
E.1 Setup

The smartphone implementation of HGO uses the python cross-platform library Kivy (`https://kivy.org`) and is optimized to deploy the Android version[8] of HGO. The application works on Android version 7+, and only requires storage permission to access the smartphone local dataset. The parameter sever handles the connection and the disconnection of devices transparently, and can be hosted in a local network or at any public IP address.

E.2 Tutorials

To join the distributed training from the Android application, the application first needs to be configured (Figure 16, right side) by setting the correct IP address and port of the server (the user must make sure they are in the same network, in case of a private IP address). Next, the computation profile needs to be selected. For simplicity, we suggest three configurable computation profiles with the following default values:

[8]The application is cross-platform and can also be deployed on IOS, Linux, MacOS and Windows.



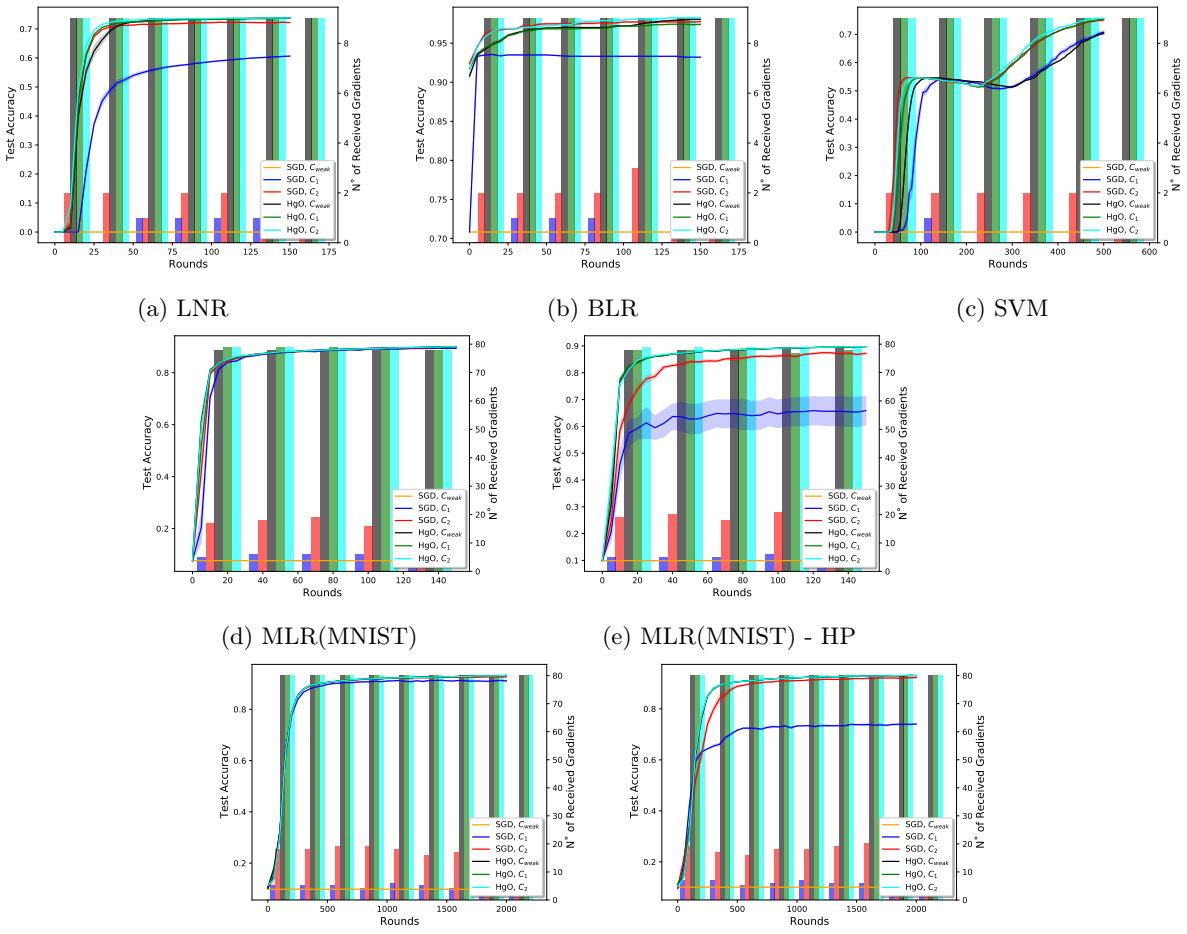
(a) MLR(MNIST) (b) MLR(MNIST) - HP

(c) FF-NN (d) FF-NN - HP

Figure 13: Analyzing the accuracy against the actual runtime of both algorithms.

- Low configuration: implying a low computation rate.
- Moderate configuration: implying an average computation rate.
- Powerful configuration: no assumption on the computation rate

Once configured, the application will receive the selected model from the PS. The PS will wait for enough devices to join. Then, it sends the current model parameters and the selected block to train in the next round. Once the training is finished, a new screen is activated with a summary of the training.

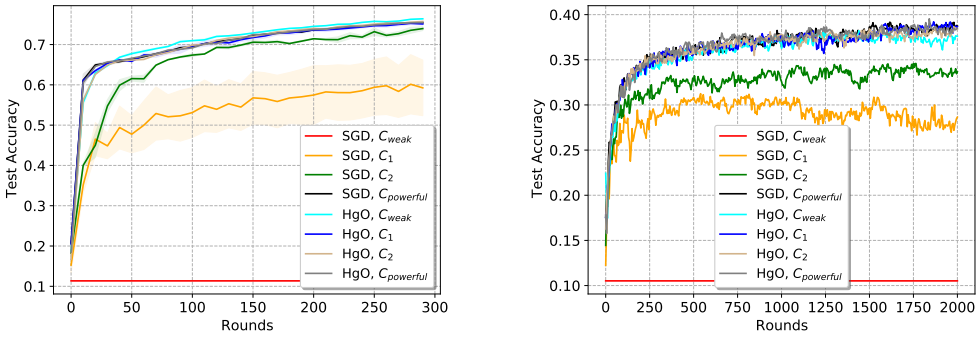


(a) LNR (b) BLR (c) SVM

(d) MLR(MNIST) (e) MLR(MNIST) - HP

(f) FF-NN (g) FF-NN.- HP

Figure 14: Analyzing the number of received gradients using both algorithms, under homogeneous and heterogeneous data partitioning.



(a) MLR(FASHION-MNIST) - HP

(b) MLR(CIFAR-10) - HP

Figure 15: Evaluating HgO on two more datasets. Note that the accuracy is low when using multinomial logistic regression on CIFAR-10. This dataset requires more complex machine learning models like convolutional neural network, in order to achieve a descent accuracy. We only include it to compare the relative performance of HgO vs SGD.

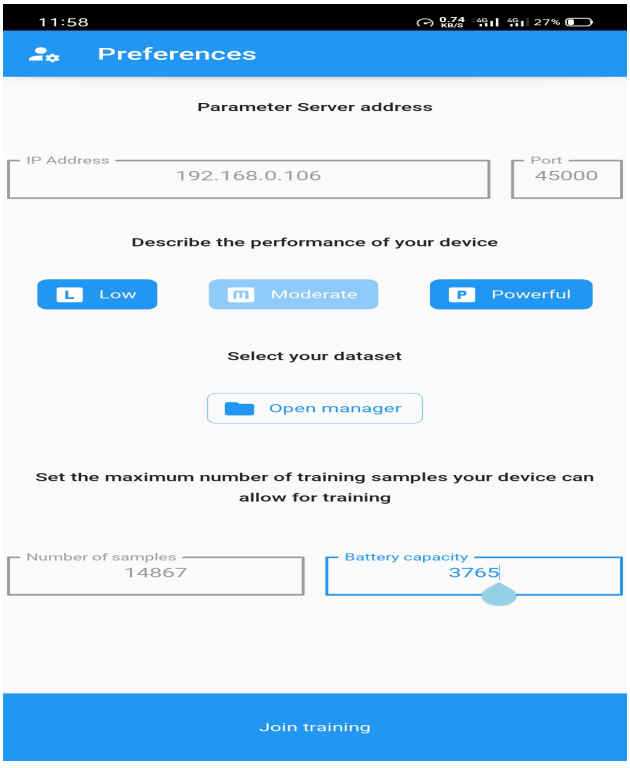


Figure 16: HgO Configuration screen on an Android smartphone.

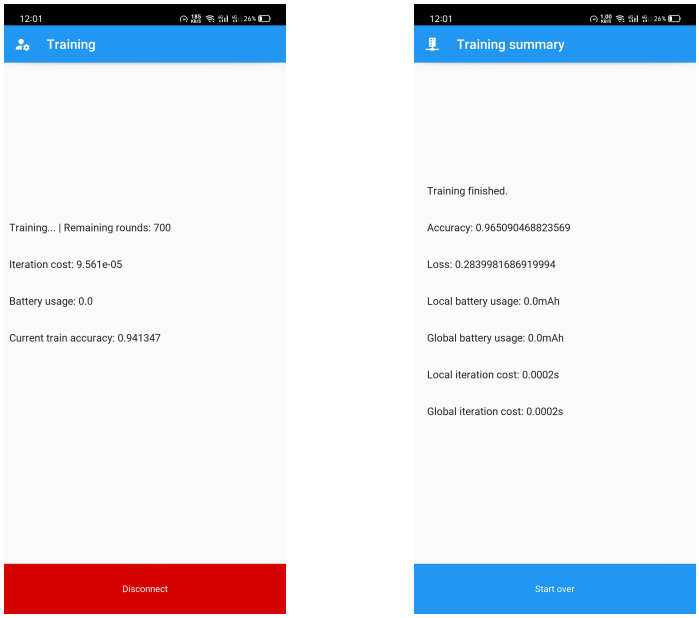


Figure 17: HGO running on an Android smartphone. The left side figure shows the app status during the training, and the right side figure shows the final screen, when the training is complete.