Characterizing & Finding Good Data Orderings for Fast Convergence of Sequential Gradient Methods

Amirkeivan Mohtashami
EPFL

Sebastian U. Stich
CISPA

Martin Jaggi
EPFL

Abstract

While SGD, which samples from the data with replacement is widely studied in theory, a variant called Random Reshuffling (RR) is more common in practice. RR iterates through random permutations of the dataset and has been shown to converge faster than SGD. When the order is chosen deterministically, a variant called incremental gradient descent (IG), the existing convergence bounds show improvement over SGD but are worse than RR. However, these bounds do not differentiate between a good and a bad ordering and hold for the worst choice of order. Meanwhile, in some cases, choosing the right order when using IG can lead to convergence faster than RR. In this work, we quantify the effect of order on convergence speed, obtaining convergence bounds based on the chosen sequence of permutations while also recovering previous results for RR. In addition, we show benefits of using structured shuffling when various levels of abstractions (e.g. tasks, classes, augmentations, etc.) exists in the dataset in theory and in practice. Finally, relying on our measure, we develop a greedy algorithm for choosing good orders during training, achieving superior performance (by more than 14 percent in accuracy) over RR.

1 Introduction

Variants of Gradient Descent are widely used for optimization of machine learning models over a dataset [5, 31]. This can be modeled as the finite-sum minimization problem

$$f^* := \min_{x \in \mathbb{R}^d} \left[ f(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x) \right].$$

In practice, computing a full gradient is expensive and therefore the gradient of individual functions $f_i$ (or a mini-batch of them) is used. While it is possible to sample this function uniformly at random, it has been observed [6, 7, 23] that traversing a (possibly random) permutation of the functions works better in practice. Recent theoretical works confirmed this observation, showing that using a random permutation instead of random sampling can lead to faster convergence [17, 20, 22, 24, 33].

In this variant of gradient descent, called Random Reshuffling (RR), a new random permutation is chosen each time one pass over the functions is completed. However, the improved convergence holds even when using the same permutation during training instead of choosing a new random one each epoch [17, 20]. This approach is called single shuffling when the permutation is chosen at random, or incremental gradient descent (IG) when the permutation is chosen deterministically. Still, the rates obtained for these two variants are worse than the one obtained for random reshuffling, the gap depending on the number of functions $N$. However, these bounds hold even for the worst choice of orderings. Meanwhile, examples exists where choosing the right order can lead to faster convergence than even Random Reshuffling [21]. Therefore, a convergence bound depending on the choice of order is needed and missing from literature. More importantly, it is currently not clear how to compare two orders in terms of their effect on convergence speed which can be useful for designing an order selection algorithm.

1CISPA Helmholtz Center for Information Security
In this work, we address this gap by introducing a measure for quantifying the suitability of each order of functions for the next epoch. We obtain upper bounds on convergence in the general non-convex setting, yielding convergence bounds tailored to any sequence of dataset orderings, while recovering and unifying the previous bounds on IG and RR. In addition, we allow access to the gradient of individual functions through a noisy oracle. To understand why this is important, note that $f_i$ can correspond to an individual data point but also may correspond to larger entities such as a worker in federated learning settings [11, 16]. In this case, even computing the gradient of individual functions is expensive. By allowing a noisy oracle for the gradient, we facilitate using an approximate version of the gradient, for example by computing it over a single mini-batch instead of the whole part of dataset available to the worker.

In addition to providing rigorous theoretical bounds, our measure also is useful in practice since it can be used in order to select orders more intelligently than randomly picking a permutation. To demonstrate this, by looking for orders that minimize our measure, we design a heuristic algorithm that can be used for choosing the right order in the next epoch. We showcase the effectiveness of our algorithm in practice by using it for training a neural network on an image classification task, namely CIFAR10 dataset. We show that our algorithm can even outperform random reshuffling in certain scenarios, improving the accuracy by more than 14 percent.

Choosing orders deterministically or uniformly at random are two extremes of the spectrum. A middle ground is randomly selecting the permutation from a more limited set of permutation. While we allow a noisy oracle for $\nabla f_i$ in general, in certain scenarios this noise might be more structured. Motivated by federated learning settings [11], where $f_i$ corresponds to $i$-th worker which itself is the average of loss for data points in the $i$-th worker, we consider the case when each $f_i$ is itself sum of a finite set of functions. We show that in this case, shuffling the top-level functions ($f_i$) followed by shuffling the low-level functions, a structure we refer to as two-level shuffling, can be beneficial both in theory and in practice.

In summary, our contributions are:

- Quantifying the effect of a chosen order on convergence, providing a unified framework that yields order specific convergence bounds while unifying previous bounds on IG and RR.
- Proposing a greedy method for finding a good order during training and establishing its usefulness in practical settings of deep network training, proving the existence of practically feasible algorithms that can beat random-reshuffling.
- Showing the possibility of intelligent random order selection by proving the benefits of choosing the permutation randomly from a limited set of permutations having a certain structure both in theory and in practice.

2 Related Works

Several previous work considered the effect of order on training. Shah et al. [25] suggests selecting a sample with the smallest loss from a set of data points sampled with replacement to make the training more robust. On the other end of the spectrum, Shumailov et al. [28] show that an adversary can slow down the training by choosing a bad order of data. At the final stages of writing this paper, we discovered a simultaneous work [15] that uses similar techniques to obtain convergence bounds depending on the order of examples and find good orders in practice. However, they do not consider a noisy access to the gradient. Moreover, we further consider the effect of imposing a structure over the random permutation on the convergence.

Several recent literature also focus on obtaining both upper and lower bounds for permutation-based SGD. For strongly convex functions with smooth components having Lipschitz Hessian, Gürbüzbalaban et al. [8] show a $\mathcal{O}(\frac{1}{T^2})$ bound, where $T$ is the number of epochs, but
requires additional assumptions such as boundedness of parameters during training. Under the same conditions, HaoChen and Sra [9] obtain the bound $O\left(\frac{1}{N^2 T^2}\right)$. In a more general settings without requiring Lipschitz Hessian, Mishchenko et al. [17] and Nguyen et al. [20] obtain the bound $O\left(\frac{1}{NT^2}\right)$. Additionally, for the smooth non-convex case considered in our work, the prove the bound $O\left(\frac{1}{N^{1/3} T^{2/3}}\right)$. The bound for IG is only investigated in the worst-case order selection, leading to an inferior bound $O\left(\frac{1}{T^{2/3}}\right)$ [17].

In [24], a $\Omega\left(\frac{1}{N^{1/2} T^{2}}\right)$ lower bound is established for random-reshuffling. On the other hand, Rajput et al. [22] shows that for 1-dimensional functions with smooth Hessian, a good order exists which yields exponential convergence, beating RR. However, their proof is non-constructive and does not provide a technique for obtaining this order. When the number of dimensions are allowed to grow more than 1, Rajput et al. [22] show that any permutation-based SGD is lower bounded by $\Omega\left(\frac{1}{N^{1/3} T}\right)$ while even in 1-dimensional a lower bound $\Omega\left(\frac{1}{T^{2/3}}\right)$ holds when the components are not convex.

Combinations of variance-reduced variants of SGD and random reshuffling have also been investigated in the literature [26]. Extending these results to order-specific convergence bounds is grounds for future work.

Importance sampling [1, 12, 30], uses only a subset of the dataset in each epoch to save on computation. In curriculum learning [3], samples are divided into easy and hard classes and only easy samples are used at the beginning in order to allow the model to learn better. Here, we mostly focus our discussion on the case where the whole dataset is traversed completely in one epoch. However, our framework allows passing through a subset of the dataset as long as certain assumptions hold and can be used to analyze training methods that rely on using a subset of the dataset or in choosing a good subset to use.

3 Order-Dependent Convergence Bound

3.1 Setup

We look into optimizing the sum of $N$ functions, i.e. the optimization problem described in (1).

We assume the individual functions are $L$-smooth.

**Assumption 1** ($L$-smoothness). Each function $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ is differentiable and there exists a constant $L > 0$ such that:

$$
\|\nabla f_i(x) - \nabla f_i(y)\| \leq L \|x - y\|, \quad \forall x, y \in \mathbb{R}^d.
$$

When considering how this formulation corresponds to optimizing a model over a dataset, a common setting is to let $f_1, \ldots, f_N$ correspond to individual data points. However, they can also represent higher abstraction levels. For example, consider an image classification task where random rotations of data points are added to the dataset for augmentation purposes which is a widely used technique [27]. In this case, $f_i$ can be the expected values of the loss over various rotations of a single image. On a higher abstraction level, $f_i$ can correspond to the average of loss of all data points belonging to $i$-th class. Alternatively, $f_i$ can correspond to different workers in federated learning settings.

However, in these cases, usually computing the exact value of $f_i$ is expensive as it requires iterating over several data points. Therefore, an approximation of the function is used for computing the gradient. For example, only a single rotation of the image is used to compute the loss or the loss for one of the data points available in the worker is returned. In order to allow this behavior in our framework, we assume we can only access each function’s gradient $\nabla f_i(x)$ through a noisy oracle $\nabla F_i(x, \xi) := \nabla f_i(x) + \xi$ where $\xi$ is picked randomly according to a distribution $D_i(x)$. Similar assumption is widely used to analyze gradient descent with sampling with replacement (SGD) where a noisy oracle for the global function $f$ is assumed [2, 4]. However, when considering permutation-based SGD, to our knowledge, allowing a
Algorithm 1 Epoch-based Gradient Descent

1: for $t = 1 \ldots T$ do
2: \hspace{1em} $x_t^1 \leftarrow x_{t-1}$
3: \hspace{1em} Determine the update sequence $r_t$.
4: \hspace{1em} Let $n$ be the length of $r_t$.
5: \hspace{1em} for $i = 1 \ldots n$ do
6: \hspace{2em} $x_i^{t+1} \leftarrow x_i^t - \gamma \nabla f_{r_t,i}(x_i^t)$
7: \hspace{1em} end for
8: \hspace{1em} $x_t^t \leftarrow x_{n+1}^t$
9: end for

Examples

Random Reshuffling
Let $n = N$ and shuffle $[N]$ to obtain the sequence $r_t, r_{t,1}, r_{t,2}, \ldots, r_{t,N}$. Incremental GD
Let $n = N$ and define the sequence $r_{t,i} = i$ for $i \in [N]$.

SGD
Let $n = 1$ and pick $r_{t,0}$ randomly from $[N]$.

noisy oracle is not investigated before. In order to obtain our convergence bound, we assume $D_i(x)$ has zero-mean and bounded variance according to Assumption 2.

Assumption 2 (Zero-mean oracle noise with bounded variance). There exists constants $P$ and $\zeta$ such that $\forall x \in \mathbb{R}^d$ and $\forall i \in [N]$:

$$E_{\xi \sim D_i(x)} \|\xi_i\|_2^2 \leq \zeta^2 + P \|\nabla f(x)\|_2^2.$$ (3)

Furthermore,

$$E_{\xi \sim D_i(x)}[\xi] = 0.$$ (4)

Remark 1. It is widely common in the literature [4, 13, 18] of SGD convergence theory to assume the average oracle noise over the whole functions is bounded by $\hat{\zeta}^2 + \hat{P} \|\nabla f(x)\|_2^2$ for some $\hat{\zeta}$ and $\hat{P}$. While here we make a slightly stronger assumption by assuming the bound applies to individual oracle noise, we note that this is still an improvement over previous work on permutation-based gradient descent which did not allow a noisy oracle assuming $\zeta = P = 0$.

3.2 Epoch-Based Gradient Descent

In order to allow a unified result for various variants of gradient descent, such as random reshuffling, and incremental gradient, we introduce a generalized template which we call epoch-based gradient descent. In this template the optimization is divided into epochs. In each epoch, a sequence $r_{t,1}, r_{t,2}, \ldots, r_{t,n}$ is chosen with elements from the set $[N] = \{1, 2, \ldots, N\}$ and the gradients of $f_{r_{t,1}}, f_{r_{t,2}}, \ldots, f_{r_{t,n}}$ are used, in this order, to update the parameters. The template is described in Algorithm 1 where we also provide examples of choosing this sequence to recover known algorithms such as random reshuffling but also sampling with replacement (SGD).

For the cases of random reshuffling or incremental gradient, $r$ would be a permutation of $[N]$ and therefore its length, $n$, would be equal to $N$. However, in general, $n$ does not have to be equal to the number of functions $N$. We allow this in order to also cover cases where a function is selected multiple times or not at all during an epoch. This can be useful for example when a subset of dataset is sampled at each epoch for optimization. Still, to ensure convergence we need certain guarantees that the selected subset is a good approximation of the global function $f$. This clearly holds for a permutation as the global function is exactly equal to the average of the selected subset. Here, we obtain this guarantee by making Assumption 3.

Assumption 3 (Bounded sample bias). We assume for all sequences $r$ chosen in an epoch, it holds that for all $x \in \mathbb{R}^d$:

$$\left\| \frac{1}{n} \sum_{i=1}^{n} (\nabla f_{r_{t,i}}(x) - \nabla f(x)) \right\|_2^2 \leq \frac{1}{4} \|\nabla f(x)\|_2^2.$$ (5)
Remark 2. In previous work, usually just the case of a permutation is considered. We emphasize that the assumption is satisfied when \( r \) is a permutation of \([N]\) or a combination of several permutations. In this case, we have \( \frac{1}{n} \sum_{i=1}^{n} (\nabla f_{r_i}(x) - \nabla f(x)) \|_2^2 = 0 \). Hence, we make a more relaxed assumption here.

For simplicity, we assume all sequences have the same length \( n \). However, our proofs can be extended to cover cases where the lengths vary per epoch in which case \( n \) should be equal to the maximum length of \( r \).

3.3 Quantifying the Effect of Order on Convergence

We now move to quantifying the effect of the chosen sequence \( r_t \) on convergence speed. Let us introduce the following quantity for any sequence \( r \) and any index \( 1 \leq k \leq n \) of that sequence at any \( x \in \mathbb{R}^d \):

\[
\phi_{r,k}^2(x) := \left\| \sum_{i=1}^{k} (\nabla f_{r_i}(x) - \nabla f(x)) \right\|_2^2.
\]

We now introduce the following assumption which assumes the values \( \phi_{r,k}(x_t) \) are bounded.

**Assumption 4** (Bounded sequence heterogeneity). There exists constants \( M_* \) and \( \sigma_* \) such that \( \forall t \in [T] \) and \( k \in [n] \):

\[
\phi_{r,k}^2(x_t) \leq \sigma_*^2 + k^2 M_* \|\nabla f(x_t)\|_2^2,
\]

where the number of epochs \( T \), number of steps in each epoch \( n \), the parameters at \( t \)-th epoch \( x_t \), and the order chosen in \( t \)-th epoch \( r_t \) are defined as in Algorithm 1.

**Remark 3.** When there is randomness in choosing \( r_t \), the above assumption can be changed to hold for the expectation over \( r_1, r_2, \ldots, r_T \). In this case, the convergence bounds we will obtain will also hold in expectation.

In the following, we propose using \( \sigma_* \) as a measure of the effect of order on convergence speed.

We note that Assumption 4 can be seen as a replacement for Assumption 5 which is commonly used in the literature \([4, 13, 29]\). Note that we use the term heterogeneity to describe the variance between different functions while we use the term noise to describe the unstructured variance of oracle’s output. Since both these terms refer to a variance, assumptions on their boundedness have a similar template.

**Assumption 5** (Bounded dataset heterogeneity). We call the set \( \{g_1, g_2, \ldots, g_n\} \) of functions \((\sigma, M)\)-heterogenous with respect to the function \( g \) if \( \forall x \in \mathbb{R}^d \):

\[
\frac{1}{n} \sum_{i=1}^{n} \|\nabla g_i(x) - \nabla g(x)\|_2^2 \leq \sigma^2 + M \|\nabla g(x)\|_2^2. \tag{7}
\]

We assume there exists constants \( M \) and \( \sigma \) such that the set \( \{f_1, \ldots, f_N\} \) is \((\sigma, M)\)-heterogenous with respect to the function \( f \).

Note that we do not need this assumption for our main result in Theorem 1. However, we mention it here to allow comparison with prior work. We now make the following remarks about how it translates to Assumption 4:

**Remark 4.** In general, Assumption 5 yields the bound \( \frac{\phi_{r,k}^2(x_t)}{k^2} \leq \frac{n}{k} \sigma^2 + \frac{nM}{k} \|\nabla f(x)\|_2^2 \). Therefore under this assumption, Assumption 4 holds with \( \sigma_*^2 = n^2 \sigma^2 \) and \( M_* = nM \). For the case of random shuffling, Mishchenko et al. \([17, Lemma 1]\) show that in \( \mathbb{E}_{r_t}, \frac{\phi_{r,k}^2(x_t)}{k^2} \leq \frac{n^2}{k} \sigma^2 + \frac{nM}{k} \|\nabla f(x_t)\|_2^2 \) and \( \sigma_*^2 = n \sigma^2 \) and \( M_* = M \).
3.4 Main Results

We now state the following convergence bound for Algorithm 1:

**Theorem 1.** If Assumptions 1, 2, 3, and 4 are satisfied, and $\gamma < \frac{1}{8\ln(M_\star + \frac{P}{n} + 1)}$, for iterates of Algorithm 1 it holds

$$\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(x_t)\|_2^2 \leq \frac{8F_0}{nT^\gamma} + 16\gamma L\zeta^2 + 32\gamma^2 L^2 \sigma^2_\star,$$

which when carefully selecting $\gamma$ yields

$$\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(x_t)\|_2^2 \in \mathcal{O} \left( \frac{L(M_\star + \frac{P}{n} + 1)F_0}{T} + \left( \frac{\sigma_\star LF_0}{nT} \right)^\frac{2}{3} + \zeta \sqrt{\frac{LF_0}{nT}} \right).$$

The proof of this theorem closely follows the proof used for convergence of random-reshuffling in [17] but covers more generalities such as oracle noise, more relaxed assumptions such as Assumption 3 or more general ones such as allowing $M_\star > 0$ in Assumption 4. We postpone the proof to Appendix A and continue to show some immediate results of this theorem including recovering previous results.

3.5 Discussion

**Recovering Bounds on Random Reshuffling and Incremental GD** Previous work use Assumption 5 to derive bounds for random reshuffling and incremental gradient descent. In Remark 4, we established the values for $\sigma_\star$ and $M_\star$ when this assumption holds both in the general case and when choosing a random permutation. Using these values, we obtain the following bounds for random reshuffling (RR) and incremental gradient descent (IG).

$$\mathcal{O} \left( \frac{L(M_\star + \frac{P}{n} + 1)F_0}{T} + \left( \frac{\sigma_\star LF_0}{nT} \right)^\frac{2}{3} + \zeta \sqrt{\frac{LF_0}{nT}} \right), \quad \text{(RR)}$$

$$\mathcal{O} \left( \frac{L(M_\star + \frac{P}{n} + 1)F_0}{T} + \left( \frac{\sigma_\star LF_0}{nT} \right)^\frac{2}{3} + \zeta \sqrt{\frac{LF_0}{nT}} \right), \quad \text{(IG)}$$

Note that Mishchenko et al. [17] derive their bounds assuming $M = 0$ in Assumption 5 while setting $P$ and $\zeta$ in Assumption 2 to zero. In this case the convergence bound for non-convex smooth case in [17] is recovered for both algorithms.

**Comparison with SGD** We use SGD to refer to sampling with replacement. Consider minimizing a finite-sum problem with a single function $f_1$ having a noisy oracle that returns the noisy gradient for one of $h_1, \ldots, h_N$ randomly. This is equivalent to running SGD over $h_1, \ldots, h_N$. Similar to previous work, we assume Assumption 5 holds for $h_i$. This means Assumption 2 holds for the oracle of $f_1$ with $\zeta = \sigma$ and $P = M$. Also, since there is only a single function $f_1$ we have $\sigma_\star = M_\star = 0$. Using these constants in Theorem 1 recovers previous results. Having a framework that can cover both random reshuffling and SGD captures the trade-off between using these methods. For example, it is also possible to consider dividing the functions to two groups, alternating between the two groups during training while each time picking a random function from the current group. We explore the usefulness of using similar schemes more in Section 5.
While this is not our main focus in this work, we would like to note that given Theorem 1, Assumption 3 provides a sufficient condition when a subset of dataset can be used with guaranteed convergence. This can be useful to analyze schemes that rely on using a subset of data. For example, using only easy samples at the beginning of curriculum learning while using the full data-set in the later stages might be justified since the easy samples might provide a good approximation when the gradient norm is large. We leave more investigation into this as a future work and return our focus to cases when a permutation of dataset is iterated at each epoch.

4 Finding Good Permutations

4.1 Effectiveness of Choosing the Right Order

In Theorem 1, the order affects the convergence speed through parameter \( \sigma_\star \). While our framework does not provide a lower bound, this still can hint that an order with a lower \( \sigma_\star \) would converge faster. While we know \( \sigma_\star \leq n^2 \sigma^2 \) when Assumption 5 holds (see Remark 4), it is not clear how good an order can be. This question allows assessing how important it is to choose the right order. To answer this question, in Appendix C, we show that \( \sigma^2_\star \in \Omega(\sigma^2) \).

Therefore, when Assumption 5 holds, the order can change the convergence speed up to a factor of \( n^2 \). In order to show that there exists cases when this lower bound is achieved, consider the example used in [24] for establishing a lower bound for RR where

\[
f_i = \begin{cases} 
\frac{x^2}{2} + \sigma x & \text{if } i \leq \frac{N}{2} \\
\frac{x^2}{2} - \sigma x & \text{if } i > \frac{N}{2}
\end{cases}.
\]

(8)

In this case, the order \( f_1, f_{\frac{N}{2}+1}, f_2, f_{\frac{N}{2}+2}, \ldots, f_{\frac{N}{2}}, f_n \), achieves \( \sigma^2_\star = \sigma^2 \). On the other hand, using Lemma 12 of [21] one can show that when choosing the order randomly, \( E \sigma^2_\star \geq \frac{\sigma^2}{256} \).

Therefore, in this case, choosing the right order strictly improves the bound in Theorem 1 over random reshuffling.

While we established the possibility of beating random reshuffling by using an order with a smaller \( \sigma_\star \), it is not clear how to find such order in the general case. We now provide a heuristic algorithm to find such orders by trying to minimize \( \sigma_\star \).

4.2 Algorithm for Finding Good Permutations

Our algorithm aims to find an order with a low \( \sigma_\star \) for the next epoch in a greedy manner. In particular, for the next function to use in the next step of epoch \( t \), the algorithm chooses one of the remaining functions that minimizes \( \phi^2_k(x_t) \) where \( x_t \) is the parameters at the beginning of \( t \)-th epoch and \( \phi \) is defined as in Assumption 4. The algorithm is described in Algorithm 2.

Algorithm 2 Greedy Order Chooser

\begin{verbatim}
for i = 1 \ldots N do
    Compute \( g^i_t \) by querying the oracle for \( \nabla f_i(x_t) \).
end for
Compute the full gradient \( g_t := \frac{1}{N} \sum_{i=1}^{N} g^i_t \).
Let \( c_{bias} \leftarrow 0 \) \quad \triangleright \text{the current bias of the found order}
for i = 1 \ldots N do
    Let \( S \) be the subset of \([N]\) not in \( r_1, \ldots, r_{i-1} \).
    Find \( r_i := \arg \min_{i \in S} ||g^i_t - g_t + c_{bias}||^2_2 \).
    Let \( c_{bias} \leftarrow c_{bias} + g^r_i - g_t \).
end for
Return \( r \).
\end{verbatim}
We note that when this algorithm is used on the example functions (8) we considered in Section 4.1, it is able to find the right order, alternating between \( f_i \) with \( i \leq \frac{n}{2} \) and \( i > \frac{n}{2} \). We showcase the effectiveness of our algorithm in more practical settings in Section 6.1 by applying it for training a neural network on CIFAR10 image classification task.

5 Structured Shuffling

5.1 Motivation

In our setup, we allowed accessing \( \nabla f_i \) through a noisy oracle. As a result, it is possible to analyze cases where \( f_i \) represents a group of data points but the oracle computes the gradient for one of them, possibly for efficiency purposes. An example is when \( f_i \) corresponds to different augmentations of the same data point.

As an another example, consider training a model in federated learning settings with \( N \) workers where at each step only a single worker is active, chosen according to a random permutation. Assume that, similar to practice, when the worker is queried for a gradient, it returns an approximation by computing the gradient for a single data point. One option is that the worker chooses this data point independently at random in which case we can model this behavior as oracle noise similar to the case of data augmentation. We refer to this case as running SGD internally.

Alternatively, and more closer to practice, the worker can return the gradient for next data point in a random permutation over its dataset. In this case, the training goes through a permutation of the combined dataset of all workers. However, this permutation is not chosen completely at random and has a structure we call two-level shuffling.

Alternatively, if there are \( m \) data points in each worker, one can use a permutation containing \( m \) copies of each \( f_i \) for selecting the active worker in the next \( mN \) steps. This is in contrast to two-level shuffling which goes through \( m \) random permutations each containing a single copy of \( f_i \). In this case, the permutation over the combined dataset is completely random, resembling random reshuffling. We refer to this case as standard shuffling.

Note that while these structures can be observed more naturally in the federated learning settings, they can also appear on single node training when grouping the data points according to an abstraction inherent to the problem. An example is grouping the data points according to their label. In this section, we investigate the effectiveness of each of these structures. In particular, we now additionally assume that each \( f_i \) is the average of several other functions. Formally, we assume there exists function \( h_{i,j} \) for each \( i \in [N] \) and \( j \in [m] \) such that

\[
f_i(x) := \frac{1}{m} \sum_{j=1}^{m} h_{i,j}(x).
\]

For simplicity and brevity, we do not assume a noisy oracle for accessing the gradients of \( h_{i,j} \). However, obtaining the results with such noise would be trivial, using Theorem 1 and exactly the same methods we use in this section.

5.2 Two-Level Shuffling

We now define two-level shuffling in a formal way and generalize it so that each time \( f_i \) is selected, it would perform \( K \) steps using \( f_i \), each time applying the gradient for next \( h_{i,j} \). This would be similar to performing \( K \) local steps each time a worker is selected in federated learning settings we considered before. We refer to this as two-level \( K \)-shuffling. The example we considered in Section 5.1 corresponds to the case when \( K = 1 \). Algorithm 3 generates a two-level \( K \)-shuffling.

Remark 5. In Algorithm 3 and the following, we assume \( K \) is a divisor of \( m \). However, this assumption can be easily avoided and is only for simplicity. One way of avoiding this...
Algorithm 3 Two-Level K-Shuffling

Let $r^{full}$ be the final ordering and initialize it to an empty list. For each $i$, determine $r^{low}_i$, the internal update sequence of $f_i$ (e.g. a permutation of $[m]$).

for $k = 1 \ldots \frac{m}{K}$ do
  Determine an order of top-level functions, $r^{top}_k$ (e.g. a permutation of $[N]$).
  for $i = 1 \ldots N$ do
    Append the next $K$ elements in $r^{low}_{r^{top}_k,i}$ (i.e. the pairs $(r^{top}_k,i, r^{low}_{r^{top}_k,i,m(k-1)+1}) \ldots (r^{top}_k,i, r^{low}_{r^{top}_k,i,mk})$) to $r^{full}$.
  end for
end for
Return $r^{full}$.

is increasing $m$ to become a divisor of $K$ by adding additional no-op second-level functions to each $f_i$ so that when these functions are selected the algorithm does nothing (e.g. the returned gradient is zero). Also note that while the algorithm is stated here for two levels, extending it to multiple levels is trivial and can be done by applying Algorithm 3 also for obtaining the low-level orders $r^{low}$.

5.3 Comparing Two-Level Shuffling and Standard Shuffling

We assume $\{f_1, \ldots, f_N\}$ is ($\sigma_{top}, M$)-heterogeneous (as defined in Assumption 5) with respect to $f$ while for each $i \in [N]$, the set of functions $\{h_{i,1}, \ldots, h_{i,m}\}$ is ($\sigma_{low}, M$)-heterogeneous with respect to $f_i$. Since these two assumptions hold, it can be seen that the set of all functions $\{h_{1,1}, \ldots, h_{N,m}\}$ is ($\sigma_{full}, M$)-heterogeneous with respect to $f$ for some $\sigma_{full}$ where $\sigma_{full}^2 \leq \sigma_{top}^2 + \sigma_{low}^2$.

We will now analyze the effect of using two-level K-shuffling on convergence when $r^{top}$ and $r^{low}$ in Algorithm 3 are random permutations of $[N]$ and $[m]$. We can directly use Theorem 1 by noticing the following lemma:

**Lemma 2.** If $r^{full}$ is a sequence obtained from Algorithm 3 when $r^{top}$ and $r^{low}$ are random permutations,

$$\mathbb{E}\sigma^2(r^{full}) \in O(NK^2\sigma_{top}^2 + (m + N(m - K))\sigma_{low}^2).$$

We postpone the proof to Appendix B and proceed to discussing the following corollary which follows directly from combining Lemma 2 and Theorem 1.

**Corollary 3.** When $\{f_i\}$ is ($\sigma_{top}, M$)-heterogeneous with respect to $f$ while $\{h_{ij}\}$ is ($\sigma_{low}, M$)-heterogeneous with respect to $f_i$, running Algorithm 1 with orders generated from Algorithm 2 using random permutations of $[N]$ and $[m]$ for $r^{top}$ and $r^{low}$ yields the convergence bound

$$O\left( \frac{L(M_* + 1)F_0}{T} + \left( \frac{(K\sigma_{top} + \sqrt{m - K}\sigma_{low})LF_0}{\sqrt{NmT}} \right)^{\frac{2}{3}} \left( \frac{\sigma_{low}LF_0}{N\sqrt{mT}} \right)^{\frac{2}{3}} \right)$$

(9)

In contrast, standard shuffling yields

$$O\left( \frac{L(M_* + 1)F_0}{T} + \left( \frac{\sigma_{full}LF_0}{\sqrt{NmT}} \right)^{\frac{2}{3}} \right)$$

(10)

When $\sigma_{full} \in \Theta(\sigma_{top} + \sigma_{low})$, setting $K$ to $\sqrt{m}$ yields the same bound when using K-shuffling as when using simple shuffling. However, for example in federated learning, using
$K = \sqrt{m}$ reduces the communication costs by $\sqrt{m}$, making $\sqrt{m}$-shuffling a better alternative to simple shuffling. Moreover, $K$-shuffling can yield improved rates in other cases by tuning $K$. This can be especially observed for the case when either $\sigma_{\text{top}} \gg \sigma_{\text{low}}$ by setting $K = 1$ or $\sigma_{\text{top}} \ll \sigma_{\text{low}}$ by setting $K = m$ obtaining a better bound by a factor of $\sqrt{m}$ or $\sqrt{N}$ respectively. In federated learning, these correspond to doing a single local step when workers have a highly heterogeneous distribution while doing $m$ local steps when they are homogeneous.

We verify the effectiveness of using $1$-shuffling in practice in Section 6.2. Here, we continue by comparing running SGD internally, i.e. sampling with replacement from $h_{i,j}$ as defined in Section 5.1, against shuffling.

In this case, the variance in the lower level becomes part of the noise of the oracle for $f_i$s and therefore decays slower than when using shuffling. On the other hand, the upper bound on the learning rate in Theorem 1 becomes larger, allowing faster convergence. In particular, the convergence bound obtained from Theorem 1 after $NmT$ steps is

\[ O \left( \frac{L(M_s + 1)F_0}{mT} + \left( \frac{\sigma_{\text{top}} LF_0}{\sqrt{N}mT} \right)^{\frac{2}{3}} + \sigma_{\text{low}} \sqrt{\frac{LF_0}{NmT}} \right). \]

(11)

In comparison with (10), it can be seen that the first term and the term containing $\sigma_{\text{top}}$ are becoming smaller faster, while the term containing $\sigma_{\text{low}}$ decays slower. This can be preferred when the term containing $\sigma_{\text{low}}$ is much smaller than the other terms. This can especially happen at the early stages of training. In particular, when $T < \frac{L F_0 \sqrt{N}}{\sigma_{\text{low}}}$, the last term in (11) is less than both the first and last term in (9).

This discussion can point to the fact that the best convergence speed might be obtained by using a combination of these cases, possibly starting with running SGD internally with a large learning rate and gradually switching to shuffling while decaying the learning rate. This resembles the learning rate decay method widely utilized in practice [32].

6 Experiments

6.1 Finding Good Orders with Algorithm 2

According to the theoretical arguments in Section 4, it can be observed that the gap between using Algorithm 2 and random reshuffling depends on the variance of functions, $\sigma^2$. However, this value is not necessarily large enough for us to observe a large gap in practice especially due to mini-batching which further reduces the variance. In order to avoid this problem, we use a special mini-batching scheme, called same-class batching.

In same-class batching, data points belonging to each class are separately batched so each batch contains data points from the same class. Moreover, these batches are kept fixed throughout training. This is unlike the standard batching where consecutive elements of a random permutation of the dataset are batched together. The permutation is chosen again at random at the beginning of each epoch.

Note that if $f_i$ corresponds to the average of loss on data points belonging to the $i$-th class, using same-class batching with batch size $\tau$ would keep the variance between batches equal to $\sigma^2$ while using standard batching would reduce it by a factor of $\tau$.

Since batch normalization layers [10] depend on the batch average during training, we suspect that they might introduce side effects when combined with same-class batching and therefore do not use them in our experiments.

We now train a ResNet-18 without Batch Normalization on CIFAR10 [14] dataset both with and without same-class batching utilizing both random reshuffling, the standard method widely used for training neural networks, or Algorithm 2 to find a good order for the next epoch. We train the models for 200 epochs using SGD optimizer with 0.9 momentum, decaying the learning rate after epochs 80, 120, and 160 by a factor 0.1. The
Table 1: Test accuracies at the end of training when using RR and Algorithm 2 (denoted by Greedy) with standard or same-class batching. Slow Update refers to the case when the order is only updated (using Algorithm 2) every 10 epochs. The average of 3 runs is reported with the standard error stated inside parantheses.

|                | Same-Class | Standard |
|----------------|------------|----------|
| RR             | 92.72(0.16)| 73.46(0.34)|
| Greedy         | 92.67(0.54)| 88.43(0.07)|
| Slow Update    | 89.51(0.72)| 83.62(0.52)|

initial learning rate is set to 0.1 for standard batching and to 0.01 for same-class batching. The experiments are repeated 3 times with different random seeds. The test accuracies at the end of training are reported in Table 1. The trajectory of test accuracy during training is also plotted in Appendix D. While the Greedy algorithm is able to obtain similar accuracy to RR with standard batching, it significantly outperforms RR when same-class batching is used.

Algorithm 2 requires computing a full gradient which is computationally expensive. Therefore, we additionally consider changing the order every 10 epochs instead of at every epoch. This leads to a small drop in accuracy but reduces the computation cost noticeably.

6.2 Two-Level Shuffling

We established that using two level shuffling, better convergence bounds can be obtained under mild conditions. In order to show that this is also true in practice, we compare the performance of standard shuffling and two-level shuffling on a set of functions for which we can control $\sigma_{top}$ and $\sigma_{low}$. We define the global function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to be

$$f(x) := \frac{1}{2} \langle Ax, x \rangle + \frac{\lambda}{2} \|x\|^2,$$

with $d = 20$, $\lambda = 0.2$, and band-diagonal matrix $A$ with $[-1_{d-1}, 21_d, -1_{d-1}]$ on the diagonals. This is a challenging problem without regularization [19]. We now define two top-level functions $f_1$ and $f_2$, and subsequently $h_{i,j}$ to be:

$$f_1(x) := f(x) + \sigma_{top} \langle 1, x \rangle$$
$$f_2(x) := f(x) - \sigma_{top} \langle 1, x \rangle$$
$$h_{i,j}(x) := \begin{cases} f_i(x) + \sigma_{low} \langle 1, x \rangle & \text{if } j \leq \frac{m}{2} \\ f_i(x) - \sigma_{low} \langle 1, x \rangle & \text{if } j > \frac{m}{2} \end{cases}$$

We set $\sigma_{low} = 10$ and compare the number of steps required to obtain accuracy $\|x\|_2 < 0.2$ for different values of $\sigma_{top}$ and $m$, the number of functions corresponding to each $f_i$. For each pair and each order generating method, we tune the learning rate over the grid $1.1 \cdot \{2^{-1}, \ldots, 2^{-20}\}$ based on performance over 3 runs. The result is plotted in Figure 1. It can be observed that as the value of $\sigma_{top}$ grows, two-level shuffling clearly outperforms simple shuffling, confirming our theoretical arguments in practice.

7 Future Work

While we have presented the efficacy of Algorithm 2, running this algorithm at epoch is costly. We also showed an alternative by running this algorithm every few epochs but observed a small impact on accuracy. Finding efficient algorithms for finding good orders obtaining the same accuracy is therefore grounds for future work. Exploring the effectiveness of two-level shuffling in other practical scenarios is also of interest. From a theoretical
perspective, currently there is a gap between the best lower bound $O\left(\frac{1}{N}\right)$ and the best upper bound $O\left(\frac{1}{N^2}\right)$ achievable by optimizing order in our framework. Filling this gap can also be grounds for future work.

8 Conclusion

We introduced a measure for quantifying the effect of order on convergence of gradient descent, establishing a framework that yields a convergence bound tailored to any custom ordering. By designing an algorithm for selecting good orders and showing its effectiveness in practice, we exemplified the practical usefulness of our framework. Finally, we demonstrated that using structured shuffling can be beneficial, proposing two-level shuffling and showing its superiority in theoretical and practical settings.

References

[1] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

[2] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *31st International Conference on Algorithmic Learning Theory (ALT)*, volume 117, pages 111–132. PMLR, 2020.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML ’09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

[4] L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. doi: 10.1137/16M1080173. URL https://doi.org/10.1137/16M1080173.
[5] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010.

[6] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[7] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. Towards a unified architecture for in-rlbms analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 325–336, 2012.

[8] Mert Gürbüzbalaban, Asu Ozdaglar, and Pablo Parrilo. Why Random Reshuffling Beats Stochastic Gradient Descent. *Mathematical Programming*, 186(1-2):49–84, March 2021. ISSN 0025-5610, 1436-4646. doi: 10.1007/s10107-019-01440-w. URL [http://arxiv.org/abs/1510.08560](http://arxiv.org/abs/1510.08560). arXiv: 1510.08560.

[9] Jeff Z. HaoChen and Suvrit Sra. Random Shuffling Beats SGD after Finite Epochs. *arXiv:1806.10077 [math, stat]*, October 2019. URL [http://arxiv.org/abs/1806.10077](http://arxiv.org/abs/1806.10077). arXiv: 1806.10077.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

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

[12] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pages 2525–2534. PMLR, 2018.

[13] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[14] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[15] Yucheng Lu, Si Yi Meng, and Christopher De Sa. A general analysis of example-selection for stochastic gradient descent. In *International Conference on Learning Representations, 2022*. URL [https://openreview.net/forum?id=7gWS3rF3opB](https://openreview.net/forum?id=7gWS3rF3opB).

[16] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguería y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[17] Konstantin Mishchenko, Ahmed Khaled Ragab Bayoumi, and Peter Richtárik. Random reshuffling: Simple analysis with vast improvements. *Advances in Neural Information Processing Systems*, 33, 2020.

[18] Amirkeivan Mohtashami, Martin Jaggi, and Sebastian U Stich. Simultaneous training of partially masked neural networks. *arXiv preprint arXiv:2106.08895*, 2021.

[19] Yurii Nesterov. *Introductory Lectures on Convex Optimization*, volume 87 of *Springer Science & Business Media*. Springer US, Boston, MA, 2004.
[20] Lam M. Nguyen, Quoc Tran-Dinh, Dzung T. Phan, Phuong Ha Nguyen, and Marten van Dijk. A unified convergence analysis for shuffling-type gradient methods. *Journal of Machine Learning Research*, 22(207):1–44, 2021.

[21] Shashank Rajput, Anant Gupta, and Dimitris Papailiopoulos. Closing the convergence gap of SGD without replacement. *arXiv:2002.10400 [cs, math, stat]*, July 2020. URL http://arxiv.org/abs/2002.10400. arXiv: 2002.10400.

[22] Shashank Rajput, Kangwook Lee, and Dimitris Papailiopoulos. Permutation-Based SGD: Is Random Optimal? *arXiv:2102.09718 [cs, math, stat]*, November 2021. URL http://arxiv.org/abs/2102.09718. arXiv: 2102.09718.

[23] Benjamin Recht and Christopher Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[24] Itay Safran and Ohad Shamir. How good is SGD with random shuffling? In *Conference on Learning Theory*, pages 3250–3284. PMLR, 2020.

[25] Vatsal Shah, Xiaoxia Wu, and Sujay Sanghavi. Choosing the sample with lowest loss makes sgd robust. In *International Conference on Artificial Intelligence and Statistics*, pages 2120–2130. PMLR, 2020.

[26] Ohad Shamir. Without-Replacement Sampling for Stochastic Gradient Methods: Convergence Results and Application to Distributed Optimization. *arXiv:1603.00570 [cs, math, stat]*, October 2016. URL http://arxiv.org/abs/1603.00570. arXiv: 1603.00570.

[27] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[28] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross Anderson. Manipulating sgd with data ordering attacks. *arXiv preprint arXiv:2104.09667*, 2021.

[29] Sebastian Stich, Amirkeivan Mohtashami, and Martin Jaggi. Critical parameters for scalable distributed learning with large batches and asynchronous updates. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 4042–4050. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/stich21a.html.

[30] Sebastian U Stich, Anant Raj, and Martin Jaggi. Safe adaptive importance sampling. In *NeurIPS 2017 - Advances in Neural Information Processing Systems*, volume 30, 2017.

[31] Ruo-Yu Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8(2):249–294, 2020.

[32] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

[33] Chulhee Yun, Shashank Rajput, and Suvrit Sra. Minibatch vs Local SGD with Shuffling: Tight Convergence Bounds and Beyond. *arXiv:2110.10342 [cs, math, stat]*, October 2021. URL http://arxiv.org/abs/2110.10342. arXiv: 2110.10342.
A Proof of Theorem 1

Lemma 4. If Assumptions 1, 2, and 3 hold, and $\gamma < \frac{1}{2L(n+P)}$, we have

$$
\mathbb{E}_{\xi, t} f(x_{t+1}) \leq f(x_t) - \frac{\gamma n}{8} \|\nabla f(x_t)\|^2 - \frac{\gamma n}{4} \|\frac{g_t}{n}\|^2 + \gamma L^2 V_t + \frac{\gamma^2 L n}{2} \zeta^2,
$$

(12)

where $V_t = \sum_{i=1}^n \|x_t - x^*_i\|^2$.

Proof. Let $g_t := \sum_{i=1}^n \nabla f_r(x^*_i)$ and $\xi_t$ represent the sequence of noises for the $t$-th epoch.

$$
\mathbb{E}_{\xi, t} f(x_{t+1}) \leq \mathbb{E}_{\xi, t} f(x_t) + \mathbb{E}_{\xi, t} (\nabla f(x_t), x_{t+1} - x_t) + \mathbb{E}_{\xi, t} \frac{L}{2} \|x_{t+1} - x_t\|
$$

$$
= f(x_t) - \gamma n \langle \nabla f(x_t), \frac{g_t}{n} \rangle + \frac{\gamma^2 L n^2}{2} \|\frac{g_t}{n}\|^2 + \frac{\gamma^2 L}{2} \sum_{i=1}^n \|\xi_t^i\|^2
$$

$$
= f(x_t) - \gamma n \|\nabla f(x_t)\|^2 + \frac{\gamma^2 L n^2}{2} \|\frac{g_t}{n}\|^2 + \frac{\gamma^2 L}{2} \sum_{i=1}^n \|\xi_t^i\|^2
$$

$$
\leq f(x_t) - \frac{\gamma n}{2} (1 - \gamma L n) \|\nabla f(x_t)\|^2 + \frac{\gamma^2 L n^2}{2} \|\frac{g_t}{n}\|^2 + \frac{\gamma^2 L n}{2} \zeta^2
$$

where in (1) we used the independence of $\xi_t^i$. We now proceed by bounding the distance of average of updates to the true gradient:

$$
\|\nabla f(x_t) - \frac{g_t}{n}\|^2 = \|\frac{1}{n} \sum_{i=1}^n [\nabla f_r(x_t) - \nabla f_r(x^*_i)] + \frac{1}{n} \sum_{i=1}^n [\nabla f_r(x_t) - \nabla f(x_t)]\|^2
$$

$$
\leq 2 \|\frac{1}{n} \sum_{i=1}^n [\nabla f_r(x_t) - \nabla f_r(x^*_i)]\|^2 + 2 \|\frac{1}{n} \sum_{i=1}^n [\nabla f_r(x_t) - \nabla f(x_t)]\|^2
$$

$$
\leq \frac{2}{n} \sum_{i=1}^n \|\nabla f_r(x_t) - \nabla f_r(x^*_i)\|^2 + 2 \phi_n^2(x_t)
$$

$$
\leq \frac{2}{n} \sum_{i=1}^n L^2 \|x_t - x^*_i\|^2 + 2 \phi_n^2(x_t)
$$

$$
= \frac{2L^2}{n} V_t + 2 \phi_n^2(x_t)
$$

$$
\leq \frac{2L^2}{n} V_t + \frac{1}{4} \|\nabla f(x_t)\|^2
$$

Where the last inequality holds because of Assumption 3. Using $\gamma < \frac{1}{2L(n+P)}$ and applying
Lemma 5. If Assumptions 1, 2, and 4 are satisfied, and $\gamma \leq \frac{1}{\kappa(n+\Phi)}$, then:

$$\sum_{k=1}^{n} \|x_k^i - x_i\|_2^2 \leq 4\gamma^2 n^3 \|f(x_i)\|_2^2 + 4\gamma^2 n\sigma_*^2$$ \hfill (13)

Proof.

$$\|x_k^i - x_i\|_2^2 = \gamma^2 \left( \sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i)\|_2^2 + \gamma^2 \sum_{i=1}^{k-1} \|\xi_i\|_2^2 \right) \leq \gamma^2 \left( \sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i)\|_2^2 + \gamma^2 P \sum_{i=1}^{k-1} \|\nabla f(x_i)\|_2^2 + \gamma^2 (k-1)\zeta^2 \right) \leq 2\gamma^2 \left( \sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i)\|_2^2 + \gamma^2 P \sum_{i=1}^{k-1} \|\nabla f(x_i)\|_2^2 + \gamma^2 (k-1)\zeta^2 \right)$$

$$= 2\gamma^2 P \sum_{i=1}^{k-1} \|\nabla f(x_i) - \nabla f(x_i)\|_2^2 + 2\gamma^2 P \sum_{i=1}^{k-1} \|\nabla f(x_i)\|_2^2 + \gamma^2 (k-1)\zeta^2$$

$$\leq 2\gamma^2 (k-1 + P) L^2 \sum_{i=1}^{k-1} \|x_i^i - x_i\|_2^2 + 2\gamma^2 \sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i)\|_2^2 + 2\gamma^2 P(k-1)\|\nabla f(x_i)\|_2^2 + \gamma^2 (k-1)\zeta^2$$

$$\leq 2\gamma^2 (k-1 + P) L^2 \sum_{i=1}^{k-1} \|x_i^i - x_i\|_2^2 + 2\gamma^2 \sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i)\|_2^2 + 2\gamma^2 P(k-1)\|\nabla f(x_i)\|_2^2 + \gamma^2 n\zeta^2$$

In order to bound the second term we can write

$$\left( \sum_{i=1}^{k} \|\nabla f_{r_i}(x_i)\|_2^2 \right)^2 = \| (k-1) \nabla f(x_i) + \sum_{i=1}^{k-1} (\nabla f_{r_i}(x_i) - \nabla f(x_i)) \|_2^2 = 2(k-1)^2 \|\nabla f(x_i)\|_2^2 + 2\sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i) - \nabla f(x_i)\|_2^2$$

$$\leq 2(k-1)^2 \|\nabla f(x_i)\|_2^2 + 2\sum_{i=1}^{k-1} \|\nabla f_{r_i}(x_i) - \nabla f(x_i)\|_2^2$$

Putting the result back we get:

$$\|x_k^i - x_i\|_2^2 \leq 2\gamma^2 (k-1 + P) L^2 \sum_{i=1}^{k-1} \|x_i^i - x_i\|_2^2 + 4\gamma^2 (k-1)(k-1 + \frac{P}{1+M_*})(1 + M_*) \|\nabla f(x_i)\|_2^2 + 2\gamma^2 \sigma_*^2 + \gamma^2 n\zeta^2$$

Denoting $V_i := \sum_{k=1}^{n} \|x_k^i - x_i\|_2^2$ and summing up for all $k$ we get:

$$V_i \leq \gamma^2 n(n-1 + 2P) L^2 V_i + \frac{2}{3} \gamma^2 (1 + M_*) (2n + 3\frac{P}{1+M_*} - 1)n(n-1) \|\nabla f(x_i)\|_2^2 + 2\gamma^2 n\sigma_*^2 + \gamma^2 n^2\zeta^2$$

$$\leq \gamma^2 n(n-1 + 2P) L^2 V_i + \frac{2}{3} \gamma^2 (1 + M_*) (2n + 3\frac{P}{1+M_*} - 1)n(n-1) \|\nabla f(x_i)\|_2^2 + 2\gamma^2 n\sigma_*^2 + \gamma^2 n^2\zeta^2$$
Therefore, applying \( \gamma < \frac{1}{2L(n+P)} \), we get \( \gamma^2 n(n - 1 + 2P)L^2 < \frac{1}{2} \) which yields

\[
V_t \leq \frac{4}{3} \gamma^2 (1 + M_*) (2n + 3 \frac{P}{1 + M_*} - 1) n(n - 1) \| \nabla f(x_t) \|_2^2 + 4 \gamma^2 n \sigma^2_* + 2 \gamma^2 n^2 \zeta^2
\]
\[
\leq \frac{4}{3} \gamma^2 n^2 (n + nM_* + P) \| \nabla f(x_t) \|_2^2 + 4 \gamma^2 n \sigma^2_* + 2 \gamma^2 n^2 \zeta^2.
\]

**Proof of Theorem 1.** Using the bound of Lemma 5 to the result of Lemma 4 and applying \( \gamma < \frac{1}{8L(n+nM_*+P)} \) we get

\[
E \xi_t f(x_{t+1}) \leq f(x_t) - \frac{\gamma n}{12} \| \nabla f(x_t) \|_2^2 + \gamma L \zeta^2 + 4 \gamma^2 n \sigma^2_*.\]

Rearranging we get:

\[
\frac{1}{12} \| \nabla f(x_t) \|_2^2 \leq \frac{f(x_t) - E \xi_t f(x_{t+1})}{\gamma n} + \gamma L \zeta^2 + 4 \gamma^2 n \sigma^2_*.\]

We now denote \( F_0 := F(x_0) - f^* \) and take the average over all the steps while taking the expectation over \( \xi_t \) for all \( t \). This yields:

\[
\frac{1}{12T} \sum_{i=1}^T \| \nabla f(x_t) \|_2^2 \leq \frac{F_0}{\gamma n T} + \gamma L \zeta^2 + 4 \gamma^2 n \sigma^2_* \]

Finding a good learning rate using Lemma 17 of [13] we get:

\[
O \left( \sqrt{L F_0 / n T} + \left( \frac{\sigma_* L F_0}{n T} \right)^\frac{3}{2} + \frac{L (M_* + \frac{P}{n} + 1) F_0}{T} \right).\]

**B  Proof of Lemma 2**

**Proof.** For brevity, let us misuse the notation and right \( h_{r_{i,j}^{top},r_{i,j}^{low}} \) instead of \( h_{r_{i,j}^{top},r_{i,j}^{low}} \).

Also, let us define \( r_0 \) and \( r_1 \) such that \( r_{i}^{full} =: (r_{0i}, r_{1i}) \). We can now write:
\begin{align*}
\mathbb{E}_{r^\text{full}} \phi^2_{r^\text{full}, (aN+b)K+c}(x) &= \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f(x) \right\|^2_2 \\
& \overset{(A)}{=} \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f_{r0i}(x) \right\|^2_2 + \mathbb{E}_{r^\text{top}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla f_{r0i}(x) - \nabla f(x) \right\|^2_2 \\
& = \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f_{r0i}(x) \right\|^2_2 \\
& + \mathbb{E}_{r_{a+1,b+1}} \left\| \left( \sum_{j=1}^{b-1} \nabla f_{r_{a,j+1,i}}(x) - \nabla f_{r_{a,j+1,i}}(x) \right) + \left( \sum_{j=1}^{b-1} \nabla f_{r_{a,j+1,i}}(x) - \nabla f(x) \right) \right\|^2_2 \\
& \leq \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f_{r0i}(x) \right\|^2_2 + 2 \mathbb{E}_{r_{a+1,b+1}} \phi^2_{r_{a+1,b+1}} + \phi^2_{r_{a+1,b+1}} \\
& \leq \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f_{r0i}(x) \right\|^2_2 + 2((K-1) + 1) \mathbb{E}_{r_{a+1,b+1}} \phi^2_{r_{a+1,b+1}} \\
& \leq \mathbb{E}_{r^\text{full}} \left\| \sum_{i=1}^{(aN+b)K+c} \nabla h_{r^\text{full}}(x) - \nabla f_{r0i}(x) \right\|^2_2 + 2NK^2 \sigma^2_{\text{top}} \\
& \overset{(B)}{=} \sum_{i=1}^{b} \mathbb{E}_{r^\text{full}} \phi_{r_{a+1,i}}(aK) + \mathbb{E}_{r^\text{full}} \phi_{r_{a+1,i}}(aK) + \sum_{i=b+2}^{N} \mathbb{E}_{r^\text{full}} \phi_{r_{a+1,i}}(aK) + 2NK^2 \sigma^2_{\text{top}} \\
& \leq \sigma^2_{\text{low}} (b(a+1)K(m - (a+1)K) + \frac{(aK+c)(m - (aK+c))}{m-1} + \frac{(N-b-1)(aK(m - aK))}{m-1}) \]
\end{align*}

where in (A) and (B) we used the randomness of \( r_{\text{low}} \) and \( r_{\text{top}} \). (C) can be obtained by separating the cases when \( a = 0 \) and \( a \geq 1 \) and noticing \( a \leq \frac{m}{K} - 1 \), \( b \leq N - 1 \) and
\[ c \leq K - 1. \]

\section*{C Lower bound on \( \sigma_* \)}

Using Assumption 4, we can write:

\[
\| \nabla f_r(x) - \nabla f(x) \|_2^2 = \left\| \sum_{i=1}^k (\nabla f_{r_i}(x) - \nabla f(x)) - \sum_{i=1}^{k-1} (\nabla f_{r_i} - \nabla f(x)) \right\|_2^2 \\
\leq 2\phi^2_{r,k} + 2\phi^2_{r,k-1} \\
\leq 4\sigma^2_* + 4k^2 M_* \| \nabla f(x) \|_2^2 \\
\leq 4\sigma^2_* + 4n^2 M_* \| \nabla f(x) \|_2^2
\]

This means that Assumption 5 should hold for some \( \sigma^2 \leq 4\sigma^2_* \) which in turn means \( \sigma_* \in \Omega(\sigma^2) \).

\section*{D Test Accuracy During Training With and Without Algorithm 2}

![Test Accuracy Plot]

Figure 2: Test accuracy plot of using random-reshuffling and the greedy algorithm with same-class batching. Greedy refers to Algorithm 2 while RR refers to Random Reshuffling. Algorithm 2 clearly outperforms RR even when the order is only updated every 10 epochs (denoted by Slow Update).