Stochastic Reweighted Gradient Descent

Ayoub El Hanchi [1] **David A. Stephens** [1]

Abstract

Despite the strong theoretical guarantees that variance-reduced finite-sum optimization algorithms enjoy, their applicability remains limited to cases where the memory overhead they introduce (SAG/SAGA), or the periodic full gradient computation they require (SVRG/SARAH) are manageable. A promising approach to achieving variance reduction while avoiding these drawbacks is the use of importance sampling instead of control variates. While many such methods have been proposed in the literature, directly proving that they improve the convergence of the resulting optimization algorithm has remained elusive. In this work, we propose an importance-sampling-based algorithm we call SRG (stochastic reweighted gradient). We analyze the convergence of SRG in the strongly-convex case and show that, while it does not recover the linear rate of control variates methods, it provably outperforms SGD. We pay particular attention to the time and memory overhead of our proposed method, and design a specialized red-black tree allowing its efficient implementation. Finally, we present empirical results to support our findings.

1. Introduction

We are interested in the unconstrained minimization problem:

$$\min_{x \in \mathbb{R}^d} F(x) := \mathop{\mathbb{E}}_{\xi \sim P} \left[f(x, \xi) \right] \tag{1}$$

where ξ is a real random vector with distribution P. In machine learning, x represents the parameters of the prediction rule, ξ represents the pair of input/output variables, f the loss function, and F the risk.

In particular, we focus on the sample average approximation

---
[1]Department of Mathematics and Statistics, McGill University, Montreal, Canada. Correspondence to: Ayoub El Hanchi <ayoub.elhanchi@mail.mcgill.ca>.

to (1):

$$\min_{x \in \mathbb{R}^d} F_n(x) := \frac{1}{n} \sum_{i=1}^{n} f(x, \xi_i) \tag{2}$$

where $(\xi_i)_{i=1}^{n}$ are i.i.d. samples from P. This is also known as empirical risk minimization in machine learning.

Defining $f_i(x) := f(x, \xi_i)$ for $i \in [n]$, and letting i be a uniformly distributed random variable on $[n]$, we can rewrite (2) as:

$$\min_{x \in \mathbb{R}^d} F(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x) = \mathbb{E}\left[f_i(x) \right] \tag{3}$$

where the expectation is taken with respect to i.

We make the following assumptions on the function F and the component functions f_i:

Assumption 1. *The function $F : \mathbb{R}^d \to \mathbb{R}$ is differentiable and μ-strongly convex, that is $\forall x, y \in \mathbb{R}^d$:*

$$F(y) \geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\mu}{2} \| y - x \|_2^2$$

Assumption 2. *For all $i \in [n]$, the functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are differentiable and convex, that is, $\forall x, y \in \mathbb{R}^d$:*

$$f_i(y) \geq f_i(x) + \langle \nabla f_i(x), y - x \rangle$$

Assumption 3. *For all $i \in [n]$, the functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are L-smooth, that is, $\forall x, y \in \mathbb{R}^d$:*

$$\| \nabla f_i(y) - \nabla f_i(x) \|_2 \leq L \| y - x \|_2$$

Note that by Assumption 3 and the triangle inequality, F is also L-smooth, and we define its condition number by $\kappa = L/\mu$.

Let x^* be the unique minimizer of F. Our goal is to design the fastest algorithm that outputs a point $x \in \mathbb{R}^d$ such that $\| x - x^* \|_2^2 < \varepsilon$ for a given accuracy $\varepsilon > 0$. Following (Agarwal & Bottou, 2015), we assume that we have access to a gradient oracle that takes as input a point $x \in \mathbb{R}^d$ and an index $i \in [n]$ and returns $\nabla f_i(x)$. We then measure

the complexity of an algorithm by counting the number of oracle calls needed to achieve a desired accuracy.

The most straightforward way to solve (3) is to ignore the particular structure of F, and simply run gradient descent or accelerated gradient-descent (Nesterov, 1983) on F. Under our assumptions and definition of complexity, gradient descent converges at the rate of $O(n\kappa \log(1/\varepsilon))$, while accelerated gradient descent converges at the rate of $O(n\sqrt{\kappa} \log(1/\varepsilon))$ (Nesterov, 2018). When n is large, this becomes prohibitively expensive.

One solution to this is to view problem (3) in its expectation form, and use the stochastic approximation of gradient descent. This yields stochastic gradient descent (SGD) which estimates $\nabla F(x)$ by $\nabla f_i(x)$ for i uniformly distributed on $[n]$ in the gradient descent update (Robbins & Monro, 1951; Nemirovsky & Yudin, 1983; Nemirovski et al., 2009; Moulines & Bach, 2011). The complexity of SGD under our assumptions is known to have two regimes. Denote by σ^2 the variance of the gradient estimator of SGD at the minimizer x^*. For $\varepsilon > \sigma^2/L\mu$, SGD converges at the fast rate $O(\kappa \log(1/\varepsilon))$, while for $\varepsilon \leq \sigma^2/L\mu$, SGD suffers from the sublinear rate $O(\sigma^2/\mu^2\varepsilon)$ (Needell et al., 2014; Nguyen et al., 2018; Gower et al., 2019). When σ^2 is large, SGD takes a long time to converge to even moderately accurate solutions.

While SGD does take into account the expectation form of the objective (3), it fails to adapt to the fact that the expectation is taken over a discrete distribution which gives rise to a finite-sum structure. The last decade has seen the development of many new methods that leverage this additional structure, including (but not limited to) SAG (Roux et al., 2012; Schmidt et al., 2017), SAGA (Defazio et al., 2014), SVRG (Johnson & Zhang, 2013), and SARAH (Nguyen et al., 2017). All of these methods converge at the fast linear rate of $O((n+\kappa)\log(1/\varepsilon))$. Further work in this direction led to the development of lower-bounds for the optimization of finite-sum functions under our assumptions and the oracle model of complexity (Agarwal & Bottou, 2015; Woodworth & Srebro, 2016; Arjevani, 2017; Lan & Zhou, 2018), and, similar to the deterministic case, accelerated methods have been developed that closely match them (Lin et al., 2018; Allen-Zhu, 2018; Lan et al., 2019; Song et al., 2020). The core innovation behind this set of algorithms is the design of more efficient gradient estimators using carefully designed control variates, earning them the name of variance-reduced methods.

Despite all this progress, SGD remains the algorithm of choice in practice for some machine learning problems where it empirically outperforms variance-reduced methods (Defazio & Bottou, 2019). From an optimization perspective, there are a few explanations to this observation.

One is the phenomenon of interpolation (Ma et al., 2018; Vaswani et al., 2019a;b): if $\sigma^2 = 0$, then SGD converges linearly for any $\varepsilon > 0$ without any variance reduction. A second possible explanation is that in many cases we are content with a low accuracy solution $\varepsilon > \sigma^2/L\mu$, in which case SGD converges linearly as well. In either of these scenarios, SGD and variance-reduced methods have the same iteration complexity, but variance-reduced methods require on average three times as many gradient evaluations per iteration (SVRG/SARAH).

While these reasons can be used to explain the superior performance of SGD in some settings, there is evidence that there are scenarios where variance-reduction might be useful even when only moderate accuracy is needed. In particular, (Goyal et al., 2018) showed that reducing the variance of the gradient estimator by increasing the batch size leads to optimization gains in the training of deep neural networks for ImageNet. Even in such cases however, the user is faced with a new problem. If we use variance-reduced methods from the beginning of the optimization, we make progress three times more slowly than SGD in the initial phase, which is often the dominating phase. Ideally, we would like to pay the additional computational cost for variance-reduction only when we need it, but it is not clear how we can detect when progress becomes constrained by the variance of the gradient estimator, although some advances have been made in this direction, see for example (Pesme et al., 2020).

A pressing question that arises from our discussion is therefore: *Can we develop a variance-reduced algorithm that uses exactly the same number of gradient evaluations as SGD ?* Such an algorithm would enjoy the fast rate of SGD in the initial phase of optimization, while automatically performing variance-reduction when needed. This question is the main motivation for this paper.

Before closing this section, we give one additional motivation for our work. As previously mentioned, the core idea of variance-reduced methods is the use of carefully crafted control variates to reduce the variance of the gradient estimator. There are, however, other ways to reduce the variance of a Monte Carlo estimator. In particular, one can use importance sampling. There is a large literature on such methods. Initial attempts focused on using a fixed distribution throughout the optimization based on prior knowledge about the functions f_i (Needell et al., 2014; Zhao & Zhang, 2015; Csiba & Richtárik, 2018), and were able to show slightly favorable rates compared to uniform sampling. A more recent line of work attempts to adaptively design the distributions (Schaul et al., 2015; Loshchilov & Hutter, 2015; Alain et al., 2016; Bouchard et al., 2016; Canevet et al., 2016; Stich et al., 2017; Katharopoulos & Fleuret, 2018; Johnson & Guestrin, 2018), although the methods developed in these works mostly rely on heuristics and do

not come with theoretical guarantees. Finally, in an attempt to put adaptive importance sampling methods on a firmer theoretical ground, a recent set of papers embed the problem of designing the distributions in an online learning framework and propose methods that achieve sublinear regret (Namkoong et al., 2017; Salehi et al., 2017; Borsos et al., 2018; 2019; El Hanchi & Stephens, 2020). These analyses however are still not satisfactory as they assume that the gradient norms of the functions f_i are uniformly bounded on $\mathbb{R}^d$ which does not hold, for example, when F is strongly-convex. The second question we ask here is therefore: *Can we develop a variance-reduced algorithm based on importance sampling and which provably outperforms SGD ?*

Contributions: In this paper, we give positive answers to both questions. In particular, our contributions are as follows:

- We propose stochastic reweighted gradient descent (SRG), a variance-reduced algorithm for the optimization of finite-sums based on importance sampling.

- As oppose to SAGA/SAG which require $O(nd)$ memory, SRG only requires $O(n)$ memory.

- Unlike SVRG/SARAH which require on average three gradient evaluations per iteration, and like SGD, SRG requires a single gradient evaluation per iteration.

- We develop a specialized red-black tree that allows an efficient implementation of SRG, incurring an overhead of only $O(\log n)$ operations per iteration compared to SGD.

- We show that SRG provably outperform SGD under our assumptions. Let σ_*^2 be the minimal variance achievable through importance sampling at the minimizer x^*. Then we show that SRG has the same convergence as SGD, but with σ^2 replaced with σ_*^2, which can be up to n times smaller.

2. Algorithm

Before introducing our proposed algorithm, let us first introduce some notation. For an iteration number $k \in \mathbb{N}$, SGD performs the following update to minimize (3):

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k)$$

for a step size $\alpha_k > 0$ and a random index i_k drawn uniformly from $[n]$ and independently at each iteration. The idea behind importance sampling is to instead sample the index i_k according to a given distribution p_k on $[n]$, and to perform the update:

$$x_{k+1} = x_k - \alpha_k \frac{1}{np_k^{i_k}} \nabla f_{i_k}(x_k)$$

Algorithm 1 SRG

Parameters: step sizes $(\alpha_k)_{k=0}^\infty > 0$, lower bounds $(\varepsilon_k)_{k=0}^\infty \in (0, \frac{1}{n}]$
Initialization: $x_0 \in \mathbb{R}^d, (g_0^i)_{i=1}^n \in \mathbb{R}^d$
for $k = 0, 1, 2, \dots$ **do**
    $p_k = \arg\min_{p \in \Delta(\varepsilon_k)} \frac{1}{n^2} \sum_{i=1}^n \frac{1}{p^i} \left\|g_k^i\right\|_2^2$
    sample $i_k \sim p_k$
    $x_{k+1} = x_k - \alpha_k \frac{1}{np_k^{i_k}} \nabla f_{i_k}(x_k)$
    sample $b_k \sim \text{Bernoulli}(\frac{\varepsilon_k}{p_k^{i_k}})$
    $g_{k+1}^i = \begin{cases} \nabla f_i(x_k) & \text{if } i = i_k \text{ and } b_k = 1 \\ g_k^i & \text{otherwise} \end{cases}$
end for

---

where p_k^i is the ith component of the vector $p_k \in \Delta$, and Δ is the probability simplex in $\mathbb{R}^n$. It is immediate to verify that the importance sampling estimator of the gradient is unbiased as long as $p_k > 0$. The challenge is to design a sequence $\{p_k\}_{k=0}^\infty$ that produces more efficient gradient estimators than the ones produced by uniform sampling.

Our proposed method is a simple modification of the one given by (El Hanchi & Stephens, 2020), and is given in Algorithm 1. The algorithm follows from the following reasoning. The variance of the importance sampling gradient estimator is given by, up to an additive constant:

$$\sigma^2(x_k, p_k) := \mathop{\mathbb{E}}_{i_k \sim p_k} \left[\left\| \frac{1}{np_k^{i_k}} \nabla f_i(x_k) \right\|_2^2 \right] \qquad (4)$$
$$= \frac{1}{n^2} \sum_{i=1}^n \frac{1}{p_k^i} \left\| \nabla f_i(x_k) \right\|_2^2$$

Ideally we would like to choose $p_k \in \Delta$ so that this variance is minimized, but this is not feasible as we do not have access to the gradient norms $\|\nabla f_i(x_k)\|_2$. Inspired by control variates methods, and in particular SAG/SAGA, we instead maintain a table $(g_k^i)_{i=1}^n$ that aims to track the component gradients $(\nabla f_i(x_k))_{i=1}^n$. We then use this table to construct an approximation of the true variance:

$$\tilde{\sigma}^2(x_k, p_k) := \frac{1}{n^2} \sum_{i=1}^n \frac{1}{p_k^i} \left\|g_k^i\right\|_2^2 \qquad (5)$$

Finally, we choose p_k so as to minimize this quantity. Since this is only an approximation, we perform the minimization over the restricted simplex $\Delta(\varepsilon_k) = \{p \in \Delta \mid p \geq \varepsilon_k\}$ for a given $\varepsilon_k \in (0, 1/n)$. This enforces $p_k > 0$, while making sure the error on the approximation of the variance is taken into account. The difference between our method and the one proposed in (El Hanchi & Stephens, 2020) is in the way the table $(g_k^i)_{i=1}^n$ is updated: we add a bernoulli random variable to determine whether the update occurs or not. This

step is crucial for the analysis as we will see, although in practice it might be skipped.

The goal of the next two sections will be to (i) show that SRG can be efficiently implemented. (ii) analyse the convergence of SRG and show that it outperforms SGD.

3. Implementation

In this section, we show how Algorithm 1 can be efficiently implemented, leading to a per-iteration cost that is competitive with SGD.

First, note that while we use the table $(g_k^i)_{i=1}^n$ in the presentation of Algorithm 1, in practice it is enough to store the gradient norms $(\left\|g_k^i\right\|_2)_{i=1}^n$ only. SRG therefore requires only $O(n)$ memory compared to SAG/SAGA which require $O(nd)$ memory in general.

With memory issues out of the way, we turn to the main bottleneck in SRG which is sampling from:

$$p_k = \arg\min_{p \in \Delta(\varepsilon_k)} \frac{1}{n^2} \sum_{i=1}^n \frac{1}{p^i} \left\|g_k^i\right\|_2^2 \qquad (6)$$

The following lemma, taken from (El Hanchi & Stephens, 2020), sheds some light on the solution:

Lemma 1. *Let $\{a_i\}_{i=1}^n$ be a non-negative set of numbers and assume that there is some $i \in [n]$ such that $a_i > 0$. Let $\varepsilon \in [0, 1/n]$, and let $\pi : [n] \to [n]$ be a permutation that orders $\{a_i\}_{i=1}^N$ in a decreasing order ($a_{\pi(1)} \geq a_{\pi(2)} \geq \cdots \geq a_{\pi(N)}$). Define for $i \in [n]$:*

$$\lambda(i) := \frac{\sum_{j=1}^i a_{\pi(j)}}{1 - (n-i)\varepsilon} \qquad (7)$$

and:

$$\rho := \max\left\{i \in [n] \mid a_{\pi(i)} \geq \varepsilon\lambda(i)\right\} \qquad (8)$$

Then a solution of the optimization problem:

$$p^* = \arg\min_{p \in \Delta(\varepsilon)} \sum_{i=1}^n \frac{1}{p_i} a_i^2 \qquad (9)$$

is given by:

$$p_i^* = \begin{cases} a_i/\lambda(\rho) & \text{if } \pi(i) \leq \rho \\ \varepsilon & \text{otherwise} \end{cases} \qquad (10)$$

In the case $a_i = 0$ for all $i \in [n]$, any $p \in \Delta(\varepsilon)$ is a solution, and in the case $\varepsilon = 1/n$, $p = 1/n$ is the unique solution.

3.1. Naive implementation

To simplify notation for this section we will use the one introduced in Lemma 1, that is, we will refer to a_i instead of $\left\|g_k^i\right\|_2$.

Algorithm 2 Naive sample

> **Input:** $(a_{\pi(i)})_{i=1}^n > 0, (\lambda(i))_{i=1}^n, \varepsilon > 0$
> **Initialization:** $l \leftarrow 1, r \leftarrow n$
> **while** $l \leq r$ **do**
> $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
>   **if** $a_{\pi(m)} \geq \varepsilon\lambda(m)$ **then**
> $c \leftarrow m$
> $l \leftarrow m + 1$
>   **else**
> $r \leftarrow m - 1$
>   **end if**
> **end while**
> $\rho \leftarrow c$
> compute p^* using (10).
> sample $i \sim p^*$.
> **Output:** the index i and its probability $p^{*,i}$.

Let us for the moment assume that we have access to $(a_{\pi(i)})_{i=1}^n$, a sorted version of $(a_i)_{i=1}^n$, as well as to $(\lambda(i))_{i=1}^n$. We will worry about how to maintain these in the next subsection. How can we sample from (9) ?

The following proposition reveals a useful property, which we use to construct Algorithm 2:

Proposition 1. *With the definitions in Lemma 1, we have:*

$$a_{\pi(i)} \geq \varepsilon\lambda(i) \Rightarrow a_{\pi(l)} \geq \varepsilon\lambda(l) \quad \forall l \leq i$$
$$a_{\pi(i)} < \varepsilon\lambda(i) \Rightarrow a_{\pi(l)} < \varepsilon\lambda(l) \quad \forall l \geq i$$

Proof. The proof of the first and second statements are the same (replacing $<$ with $\geq$). We prove here the second statement. Let $i \in [n]$ such that $a_{\pi(i)} < \varepsilon\lambda(i)$. By definition of π, $a_{\pi(i+1)} \leq a_{\pi(i)}$, so we have:

$$a_{\pi(i+1)} < \varepsilon\lambda(i)$$

$$\Leftrightarrow \quad a_{\pi(i+1)}(1 - (n-i)\varepsilon) < \varepsilon \sum_{j=1}^i a_{\pi(j)}$$

$$\Leftrightarrow \quad a_{\pi(i+1)}(1 - (n-(i+1))\varepsilon) < \varepsilon \sum_{j=1}^{i+1} a_{\pi(j)}$$

$$\Leftrightarrow \quad a_{\pi(i+1)} < \varepsilon\lambda(i+1)$$

the rest of the claim holds by induction. $\square$

Lemma 2. *With the definitions of Lemma 1, Algorithm 2 samples from the distribution given by (9).*

Proof. Algorithm 2 proceeds by first finding ρ, then uses Lemma 1 to construct and sample from p^*. It is therefore enough to show that it indeed finds ρ. First, note that from

the proof of Lemma 1 in (El Hanchi & Stephens, 2020), we know that:

$$a_{\pi(1)} \geq \varepsilon\lambda(1)$$

therefore the variable c is guaranteed to be well defined by the end of the $while$ loop. We claim that the $while$ loop maintains the following invariant: $\rho \in \{c\} \cup [l,r]$ where $[l,r] = \{l, l+1, \ldots, r-1, r\}$. Therefore, at the end of the loop, we have $c = \rho$. This can easily be proved using induction. The base case is trivial as $[l,r] = [n]$ initially, and $\rho \in [n]$. The induction step follows from Proposition 1. $\qquad\square$

We now have an algorithm to sample from the distribution (6). Unfortunately, while the search for ρ only takes $O(\log n)$ operations, the computation of p^* alone takes $O(n)$ time, while naively maintaining $(\|g_k^i\|_2)_{i=1}^n$ sorted requires $O(n \log n)$ operations and updating $(\lambda(i))_{i=1}^n$ requires another $O(n)$ operations. For large enough n, this can cause significant slowdown even when the gradient evaluations are expensive.

3.2. Tree-based implementation

Our goal in this subsection is to design an efficient algorithm that allows sampling from (6) in $O(\log n)$ time. To achieve this we need to:

- Efficiently maintain $(\|g_k^i\|_2)_{i=1}^n$ sorted from one iteration to the next while allowing for fast search.

- Quickly evaluate the $(\lambda(i))_{i=1}^n$ when we need them.

- Sample from p^* without explicitly forming it.

To comply with all these requirements, we make use of an augmented red-black tree T. We will assume that for a node v, its left child $v.left$ has a smaller key than that of v, while the opposite is true for its right child $v.right$. We will refer to its parent by $v.parent$. If v has no left (or right) child, we will assume that $v.left$ (or $v.right$) takes value nil.

The keys of the nodes of the tree T are the gradient norms $(\|g_k^i\|_2)_{i=1}^n$, while their values are the corresponding indices. To simplify the presentation, we will assume that the keys are unique, although everything here still works when they are not with some small modifications. We require that each node v of the tree T maintains two more attributes:

- $v.size$ which counts the number of nodes in the subtree whose root is v.

- $v.sum$ which stores the sum of the keys of the nodes in the subtree whose root is v.

We also make use of four methods that can be efficiently implemented using these additional attributes:

- $T.rank(v)$ which returns the rank (position in the decreasing order of the tree) of the node v.

- $T.partial\_sum(v)$ which returns the sum of all keys larger than or equal to the key of the given node.

Before defining the last two methods, to each node v we associate (conceptually only) the interval $[T.partial\_sum(v) - v.key, T.partial\_sum(v))$. The last two methods are:

- $T.select\_rank(r)$ which returns the node with the rth largest key in the tree.

- $T.select\_sum(s)$ which returns the node whose associated interval contains s.

Finally, and to allow fast access to the elements $\|g_k^i\|_2$ through their indices, we store the nodes of the tree in an array whose i^{th} element is the node whose value is i. See Chapter 14 of (Cormen et al., 2009) for a detailed description of how to implement such a tree so that all the methods we require as well as the maintenance of the tree run in $O(\log(n))$ operations.

With this notation in place, we can now state our $O(\log n)$ method to sample from (6) in Algorithm 4, which uses Algorithm 3 as a subroutine.

Algorithm 3 Solve

Input: tree T, lower bound $\varepsilon > 0$
$v \leftarrow T.root$
$r \leftarrow v.right.size + 1$
$s \leftarrow v.right.sum + v.key$
$c \leftarrow 1 - (n-r)\varepsilon$
while $v \neq nil$ **do**
  if $v.key \geq \varepsilon s/c$ **then**
    $w \leftarrow v$
    $v \leftarrow v.left$
    if $v \neq nil$ **then**
      $r \mathrel{+}= v.right.size + 1$
      $s \mathrel{+}= v.right.sum + v.key$
      $c = 1 - (n-r)\varepsilon$
    end if
  else
    $v \leftarrow v.right$
    if $v \neq nil$ **then**
      $r \mathrel{-}= v.left.size + 1$
      $s \mathrel{-}= v.left.sum + v.parent.key$
      $c = 1 - (n-r)\varepsilon$
    end if
  end if
end while
Output: node w whose rank is ρ

Algorithm 4 Sample

Input: tree T, lower bound $\varepsilon > 0$
$w \leftarrow Solve(T, \varepsilon)$ {Using Algorithm 3}
$\rho \leftarrow T.rank(w)$
$\lambda \leftarrow T.partial\_sum(w)/(1 - (n - \rho)\varepsilon)$
sample u uniformly from $[0, 1)$
if $u < 1 - (n - \rho)\varepsilon$ **then**
   $s \leftarrow \lambda u$
   $v \leftarrow T.select\_sum(s)$
   $i \leftarrow v.value$
   $p^i \leftarrow v.key/\lambda$
else
   $u \;-= 1 - (n - \rho)\varepsilon$
   $r \leftarrow n - \lfloor u/\varepsilon \rfloor$
   $v \leftarrow T.select\_rank(r)$
   $i \leftarrow v.value$
   $p^i \leftarrow \varepsilon$
end if
Output: the index i and its probability p^i

Similar to Algorithm 2, Algorithm 4 first finds ρ, then uses Lemma 1 to sample from (6).

To find ρ, Algorithm 4 uses Algorithm 3, which is only superficially different from the $while$ loop in Algorithm 2. In particular, it is not hard to show that the variable r holds the rank of the node v (which corresponds to m in Algorithm 2), and the variable s and c satisfy $\lambda(r) = s/c$ (which corresponds to $\lambda(m)$ in Algorithm 2). The correctness of Algorithm 3 therefore follows from that of Algorithm 2 which we proved in the last subsection.

Once ρ is found, Algorithm 4 computes $\lambda = \lambda(p)$, and samples the index i from (6) using inverse transform sampling. In particular, it partitions the unit interval into $[0, 1 - (n - \rho)\varepsilon]$ which is reserved to the ρ largest elements, and $[1 - (n - \rho)\varepsilon, 1)$ which is reserved to the remaining $(n - \rho)$ elements all of which have probability ε. Using the methods $T.select\_sum(s)$ and $T.select\_rank(r)$, it then picks the right index and lazily evaluates the required probability. Note that all the methods we use run in $O(\log n)$ time, and therefore the total complexity of sampling from (6) and maintaining the tree T is $O(\log n)$.

4. Convergence analysis

In this section, we analyze the convergence of SRG, and show that it outperforms SGD under our assumptions. Two key constants are helpful in contrasting the convergence of SRG and SGD. Recall the definition of $\sigma^2(x_k, p_k)$ in (4),

and define:

$$\sigma^2 := \sigma^2(x^*, 1/n) = \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(x^*)\|_2^2$$

$$\sigma_*^2 := \min_{p \in \Delta} \sigma^2(x^*, p) = \frac{1}{n^2} \left(\sum_{i=1}^n \|\nabla f_i(x^*)\|_2 \right)^2$$

It is well known that the convergence of SGD depends on σ^2 (Gower et al., 2019). We here show that SRG is able to reduce this dependence to σ_*^2, which can be up to n times smaller than σ^2.

4.1. Bounding the suboptimality of p_k

We start our analysis with the following Lemma, which is a slightly improved version of Lemma 6 in (Borsos et al., 2018).

Proposition 2. *Let $\{a_i\}_{i=1}^n$ be a non-negative set of numbers and suppose that there exists an $i \in [n]$ such that $a_i > 0$. Then:*

$$\min_{p \in \Delta(\varepsilon)} \sum_{i=1}^n \frac{1}{p^i} a_i^2 \leq (1 + 2n\varepsilon) \min_{p \in \Delta} \sum_{i=1}^n \frac{1}{p^i} a_i^2$$

$$= (1 + 2n\varepsilon) \left(\sum_{i=1}^n a_i \right)^2$$

for all $0 \leq \varepsilon \leq \frac{1}{2n}$.

Proof. By Lemma 1 we have:

$$\min_{p \in \Delta(\varepsilon)} \sum_{i=1}^n \frac{1}{p^i} a_i^2 = \lambda(\rho) \sum_{i=1}^\rho a_{\pi(i)} + \sum_{i=\rho+1}^n \frac{a_{\pi(i)}^2}{\varepsilon}$$

$$\leq \lambda(\rho) \sum_{i=1}^\rho a_{\pi(i)} + a_{\pi(\rho+1)} \sum_{i=\rho+1}^n \frac{a_{\pi(i)}}{\varepsilon}$$

Where the second line follows from the definition of π. By the reverse chain of implications in the proof of Proposition 1 and taking $i = \rho$ we obtain $a_{\pi(\rho+1)} \leq \varepsilon\lambda(\rho)$. Replacing we get:

$$\min_{p \in \Delta(\varepsilon)} \sum_{i=1}^n \frac{1}{p^i} a_i^2 \leq \lambda(\rho) \sum_{i=1}^n a_i$$

$$\leq \frac{1}{1 - (n - \rho)\varepsilon} \left(\sum_{i=1}^n a_i \right)^2$$

$$\leq \frac{1}{1 - n\varepsilon} \left(\sum_{i=1}^n a_i \right)^2$$

The result then follows from the inequality $\frac{1}{1-x} \leq 1 + 2x$ valid for $x \in [0, 1/2]$. $\qquad\square$

Proposition 2 is the main motivation for our choice of p_k in Algorithm 1. If our goal was only to ensure that $p_k > 0$, we could have first performed the optimization over the entire simplex Δ, which has a simple solution $p^i \propto \left\| g_k^i \right\|_2$, and then mixed this distribution with the uniform distribution over $[n]$. This approach however does not give us any suboptimality guarantees similar to Proposition 2, which is crucial for the analysis as will soon become clear.

4.2. Useful lemmas

Before proceeding with the main results, let us first state a few useful lemmas. We refer the reader to the Appendix for their proofs. The first lemma studies the evolution of $(g_k^i)_{i=1}^n$.

Lemma 3. *Let $k \in \mathbb{N}$. Suppose $(g_k^i)_{i=1}^n$ evolves as in Algorithm 1, and assume that Assumptions 1, 2, and 3 hold. Taking expectation with respect to (i_k, b_k), conditional on $(i_t, b_t)_{t=0}^{k-1}$, we have:*

$$\mathbb{E}\left[\sum_{i=1}^n \left\| g_{k+1}^i - \nabla f_i(x^*) \right\|_2^2\right] \leq$$

$$(1 - \varepsilon_k) \sum_{i=1}^n \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + 2Ln\varepsilon_k \left[F(x_k) - F(x^*)\right]$$

The second lemma is a bound on the second moment of the gradient estimator used by SRG.

Lemma 4. *Assume $\varepsilon_k \in (0, 1/2n]$. For all $\beta, \gamma, \delta, \eta > 0$ we have:*

$$\mathbb{E}_{i_k \sim p_k}\left[\left\|\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right\|_2^2\right] \leq D_2 \frac{2L}{n\varepsilon_k}\left[F(x_k) - F^*\right] +$$

$$D_1 \frac{1}{n^2\varepsilon_k}\sum_{i=1}^n \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + D_3(1 + 2n\varepsilon_k)\sigma_*^2$$

where:

$$D_1 := (1 + \beta^{-1} + \delta) + (1 + \gamma^{-1} + \delta^{-1})(1 + \eta)$$
$$D_2 := (1 + \beta + \gamma)$$
$$D_3 := (1 + \gamma^{-1} + \delta^{-1})(1 + \eta^{-1})$$

4.3. Main results

To study the convergence of SRG, we use the following Lyapunov function, which is the same (up to constants) as the one used to study the convergence of SAGA in (Hofmann et al., 2015; Defazio, 2016).

$$T^k = T(x_k, (g_k^i)_{i=1}^n)$$

$$:= \frac{\alpha_k}{n\varepsilon_k}\frac{a}{L}\sum_{i=1}^n \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + \left\| x_k - x^* \right\|_2^2$$

for the constant $a = 0.673$ that we set during the analysis. Our main result is the following bound on the evolution of T^k along the steps of SRG which uses Lemmas 3 and 4. All the proofs of this section can be found in the Appendix.

Theorem 1. *Suppose that Assumptions 1, 2 and 3 hold, and that $(x_k, (g_k^i)_{i=1}^n)$ evolves according to Algorithm 1. Further, assume that for all $k \in \mathbb{N}$: (i) α_k/ε_k is constant. (ii) $\varepsilon_k \in (0, 1/2n]$. (iii) $\alpha_k \leq n\varepsilon_k/20L$. Then for all $k \in \mathbb{N}$:*

$$\mathbb{E}\left[T^{k+1}\right] \leq (1 - \alpha_k\zeta)\mathbb{E}\left[T^k\right] + (1 + 2n\varepsilon_k)2\alpha_k^2\sigma_*^2$$

where:

$$\zeta := \min\{L/n, \mu\}$$

The above theorem leads to the following convergence rate for SRG with a constant step size.

Corollary 1. *Under the assumptions of Theorem 1, and using a constant lower bound $\varepsilon_k = \varepsilon \in (0, 1/2n]$ and a constant step size $\alpha_k = \alpha \leq n\varepsilon/20L$, we have for any $k \in \mathbb{N}$:*

$$\mathbb{E}\left[T^k\right] \leq (1 - \alpha\zeta)^k T^0 + (1 + 2n\varepsilon)\frac{2\alpha\sigma_*^2}{\zeta}$$

Comparing this result with the standard convergence result of SGD, for example Theorem 3.1 in (Gower et al., 2019), we see that they are of similar form. The advantage of this result is the dependence on σ_*^2 instead of σ^2. This allows SRG to extend the range of accuracies on which SGD converges linearly on the one hand, and reduces the leading factor in the complexity of SGD for high accuracies on the other.

For decreasing step sizes, we have the following result, which again is similar to Theorem 3.2 in (Gower et al., 2019), but with σ_*^2 replacing σ^2.

Corollary 2. *Under the assumptions of Theorem 1, and using the step sizes and lower bounds:*

$$\alpha_k := \frac{2(k + k_0) + 1}{[c + (k + k_0)(k + k_0 + 2)]\zeta}$$

$$\varepsilon_k := \frac{20L\alpha_k}{n}$$

where:

$$k_0 := \frac{80L}{\zeta} - 2$$

$$c := \frac{40L}{\zeta}$$

Then:

$$\mathbb{E}\left[T^k\right] \leq O\left(\frac{T^0 + \sigma_*^2\log k}{k^2}\right) + O\left(\frac{\sigma_*^2}{k}\right)$$
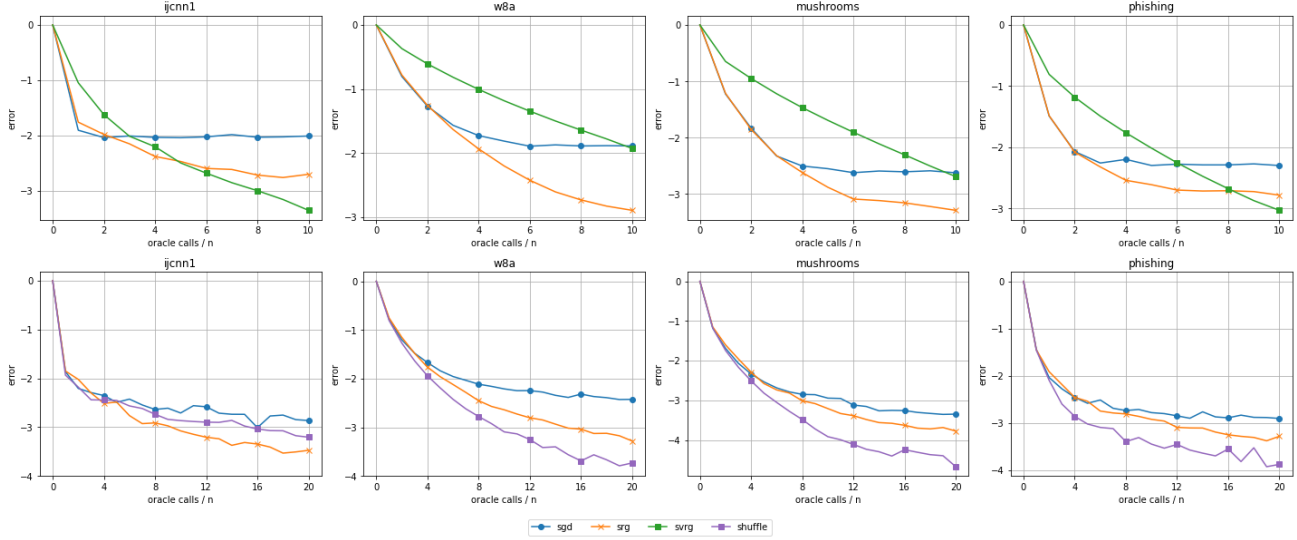
Figure 1. Comparison of the evolution of the relative error for different optimizers on ℓ_2-regularized logistic regression problems using the datasets *ijcnn1, w8a, mushrooms* and *phishing*. The top row compares SGD (blue), SRG (orange), and SVRG (green), all with constant step size. The bottom row compares SGD (blue), SRG (orange), and SGD with shuffling (purple), all using decreasing step sizes.

5. Experiments

In this section, we experiment with SRG on ℓ_2-regularized logistic regression problems. In this case, the functions f_i are given by:

$$f_i(x) := \log\left(1 + \exp\left(-y_i a_i^T x\right)\right) + \frac{\mu}{2}\|x\|_2^2$$

where $y_i \in \{0, 1\}$ is the label of data point $a_i \in \mathbb{R}^d$. It is not hard to show that each f_i is convex and $L_i = 0.25\|a_i\|_2^2 + \mu$ smooth. Their average F is also μ-strongly convex.

We experiment with four datasets from LIBSVM (Chang & Lin, 2011): *ijcnn1, w8a, mushrooms,* and *phishing*. We start by normalizing the data points $(a_i)_{i=1}^n$ so that $\|a_i\|_2 = 1$ for all $i \in [n]$. This makes all the f_i L-smooth with $L = 0.25 + \mu$. As is standard for regularized logistic regression, we take $\mu = \frac{1}{n}$. We conduct two sets of experiments: one set using a constant step size, and another using decreasing step sizes. In both cases we evaluate the performance of the algorithms by tracking the relative error $\frac{\|x_k - x^*\|_2^2}{\|x_0 - x^*\|_2^2}$. For all experiments, we sample the indices without replacement using a batch size of 128. Finally, we use the same step size across all algorithms for each dataset to fairly compare their performance. We give more details on the step sizes we use in the supplementary material.

The results of the experiments with constant step-size can be seen on the top row of Figure 1. We compare SRG with SGD on the one hand and SVRG on the other. We observe that SRG consistently outperforms SGD, reaching solutions that are about an order of magnitude more accurate. We also

notice that SRG extends the range of accuracies on which SGD dominates SVRG. Eventually, SVRG outperforms both SGD and SRG, but our results show that SRG allows reaching solutions of moderate accuracy more quickly.

The bottom row of Figure 1 shows the results of the experiments where we use decreasing step-sizes. Here we compare SRG with SGD as well as SGD with random shuffling. SGD with random shuffling has attracted a lot of attention recently, and many analyses have been developed that show its superior performance, see for example (Ahn et al., 2020). Our results confirm this as SGD with shuffling outperforms both SGD and SRG. We still see however that SRG converges slightly faster than SGD, as our analysis suggests.

While the empirical results we present here are positive but not impressive, we would like to point out that the improvement of SRG over SGD depends directly on the ratio $r = \sigma^2/\sigma_*^2$. In particular, if $\sigma_*^2 = \sigma^2$, then we should not expect any improvement. For the experiments in Figure 1, r is in the range $[5, 10]$. One can construct artificial examples in which this ratio is very large, which we do in the supplementary material, leading to significant improvements. It is however still unclear how often such cases are encountered in practice. We hope to explore this question more in future work.

References

Agarwal, A. and Bottou, L. A Lower Bound for the Optimization of Finite Sums. In *International Conference on Machine Learning*, pp. 78–86. PMLR, June 2015.

Ahn, K., Yun, C., and Sra, S. SGD with shuffling: optimal rates without component convexity and large epoch requirements. *Advances in Neural Information Processing Systems*, 33, 2020.

Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. Variance Reduction in SGD by Distributed Importance Sampling. *arXiv:1511.06481 [cs, stat]*, April 2016.

Allen-Zhu, Z. Katyusha: The First Direct Acceleration of Stochastic Gradient Methods. *Journal of Machine Learning Research*, 18(221):1–51, 2018.

Arjevani, Y. Limitations on Variance-Reduction and Acceleration Schemes for Finite Sums Optimization. *Advances in Neural Information Processing Systems*, 30: 3540–3549, 2017.

Borsos, Z., Krause, A., and Levy, K. Y. Online Variance Reduction for Stochastic Optimization. In *Conference On Learning Theory*, pp. 324–357. PMLR, July 2018.

Borsos, Z., Curi, S., Levy, K. Y., and Krause, A. Online Variance Reduction with Mixtures. In *International Conference on Machine Learning*, pp. 705–714. PMLR, May 2019.

Bouchard, G., Trouillon, T., Perez, J., and Gaidon, A. Online Learning to Sample. *arXiv:1506.09016 [cs, math, stat]*, March 2016.

Canevet, O., Jose, C., and Fleuret, F. Importance Sampling Tree for Large-scale Empirical Expectation. In *International Conference on Machine Learning*, pp. 1454–1462. PMLR, June 2016.

Chang, C.-C. and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press, July 2009.

Csiba, D. and Richtárik, P. Importance Sampling for Minibatches. *Journal of Machine Learning Research*, 19(27): 1–21, 2018.

Defazio, A. A Simple Practical Accelerated Method for Finite Sums. *Advances in Neural Information Processing Systems*, 29:676–684, 2016.

Defazio, A. and Bottou, L. On the Ineffectiveness of Variance Reduced Optimization for Deep Learning. *Advances in Neural Information Processing Systems*, 32: 1755–1765, 2019.

Defazio, A., Bach, F., and Lacoste-Julien, S. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. *Advances in Neural Information Processing Systems*, 27:1646–1654, 2014.

El Hanchi, A. and Stephens, D. Adaptive Importance Sampling for Finite-Sum Optimization and Sampling with Decreasing Step-Sizes. *Advances in Neural Information Processing Systems*, 33, 2020.

Gower, R. M., Loizou, N., Qian, X., Sailanbayev, A., Shulgin, E., and Richtárik, P. SGD: General Analysis and Improved Rates. In *International Conference on Machine Learning*, pp. 5200–5209. PMLR, May 2019.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv:1706.02677 [cs]*, April 2018.

Hofmann, T., Lucchi, A., Lacoste-Julien, S., and McWilliams, B. Variance Reduced Stochastic Gradient Descent with Neighbors. *Advances in Neural Information Processing Systems*, 28:2305–2313, 2015.

Johnson, R. and Zhang, T. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. *Advances in Neural Information Processing Systems*, 26:315–323, 2013.

Johnson, T. B. and Guestrin, C. Training Deep Models Faster with Robust, Approximate Importance Sampling. *Advances in Neural Information Processing Systems*, 31: 7265–7275, 2018.

Katharopoulos, A. and Fleuret, F. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. In *International Conference on Machine Learning*, pp. 2525–2534. PMLR, July 2018.

Lan, G. and Zhou, Y. An optimal randomized incremental gradient method. *Mathematical Programming*, 171(1): 167–215, September 2018.

Lan, G., Li, Z., and Zhou, Y. A unified variance-reduced accelerated gradient method for convex optimization. *Advances in Neural Information Processing Systems*, 32: 10462–10472, 2019.

Lin, H., Mairal, J., and Harchaoui, Z. Catalyst Acceleration for First-order Convex Optimization: from Theory to Practice. *Journal of Machine Learning Research*, 18 (212):1–54, 2018.

Loshchilov, I. and Hutter, F. Online Batch Selection for Faster Training of Neural Networks. November 2015.

Ma, S., Bassily, R., and Belkin, M. The Power of Interpolation: Understanding the Effectiveness of SGD in Modern Over-parametrized Learning. In *International Conference on Machine Learning*, pp. 3325–3334. PMLR, July 2018.

Moulines, E. and Bach, F. Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning. *Advances in Neural Information Processing Systems*, 24:451–459, 2011.

Namkoong, H., Sinha, A., Yadlowsky, S., and Duchi, J. C. Adaptive Sampling Probabilities for Non-Smooth Optimization. In *International Conference on Machine Learning*, pp. 2574–2583. PMLR, July 2017.

Needell, D., Ward, R., and Srebro, N. Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm. *Advances in Neural Information Processing Systems*, 27:1017–1025, 2014.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM Journal on Optimization*, 19(4): 1574–1609, January 2009.

Nemirovsky, A. S. and Yudin, D. B. *Problem Complexity and Method Efficiency in Optimization*. John Wiley, Chichester, 1983.

Nesterov, Y. 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.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer US, 2004.

Nesterov, Y. *Lectures on Convex Optimization*. Springer Optimization and Its Applications. Springer International Publishing, 2 edition, 2018.

Nguyen, L., Nguyen, P. H., Dijk, M., Richtarik, P., Scheinberg, K., and Takac, M. SGD and Hogwild! Convergence Without the Bounded Gradients Assumption. In *International Conference on Machine Learning*, pp. 3750–3758. PMLR, July 2018.

Nguyen, L. M., Liu, J., Scheinberg, K., and Takáč, M. SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient. In *International Conference on Machine Learning*, pp. 2613–2621. PMLR, July 2017.

Pesme, S., Dieuleveut, A., and Flammarion, N. On Convergence-Diagnostic based Step Sizes for Stochastic Gradient Descent. In *International Conference on Machine Learning*, pp. 7641–7651. PMLR, November 2020.

Robbins, H. and Monro, S. A Stochastic Approximation Method. *Annals of Mathematical Statistics*, 22(3):400–407, September 1951.

Roux, N., Schmidt, M., and Bach, F. A Stochastic Gradient Method with an Exponential Convergence _rate for Finite Training Sets. *Advances in Neural Information Processing Systems*, 25:2663–2671, 2012.

Salehi, F., Celis, L. E., and Thiran, P. Stochastic Optimization with Bandit Sampling. August 2017.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized Experience Replay. November 2015.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1):83–112, March 2017.

Song, C., Jiang, Y., and Ma, Y. Variance Reduction via Accelerated Dual Averaging for Finite-Sum Optimization. *Advances in Neural Information Processing Systems*, 33, 2020.

Stich, S. U., Raj, A., and Jaggi, M. Safe Adaptive Importance Sampling. *Advances in Neural Information Processing Systems*, 30:4381–4391, 2017.

Vaswani, S., Bach, F., and Schmidt, M. Fast and Faster Convergence of SGD for Over-Parameterized Models and an Accelerated Perceptron. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1195–1204. PMLR, April 2019a.

Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., and Lacoste-Julien, S. Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates. *Advances in Neural Information Processing Systems*, 32: 3732–3745, 2019b.

Woodworth, B. E. and Srebro, N. Tight Complexity Bounds for Optimizing Composite Objectives. *Advances in Neural Information Processing Systems*, 29:3639–3647, 2016.

Zhao, P. and Zhang, T. Stochastic Optimization with Importance Sampling for Regularized Loss Minimization. In *International Conference on Machine Learning*, pp. 1–9. PMLR, June 2015.

Supplementary material

A. Standard results

Lemma 5. *Under Assumptions 2 and 3, we have:*

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x) - \nabla f_i(x^*)\|_2^2 \leq 2L\left[F(x) - F(x^*)\right]$$

Proof. By Assumptions 2 and 3, and Theorem 2.1.5 in (Nesterov, 2004) we have for all $i \in [n]$:

$$f_i(x) \leq f_i(x^*) + \langle \nabla f_i(x^*), x - x^* \rangle + \frac{1}{2L}\|\nabla f_i(x) - \nabla f_i(x^*)\|_2^2$$

Rearranging gives:

$$\|\nabla f_i(x) - \nabla f_i(x^*)\|_2^2 \leq 2L\left[f_i(x) - f_i(x^*) - \langle \nabla f_i(x^*), x - x^* \rangle\right]$$

Averaging over $i \in [n]$ and noticing that $\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x^*) = \nabla F(x^*) = 0$, we get the result. $\qquad\square$

Proposition 3. *(Young's inequality, Peter-Paul inequality) Let $a, b, c \in \mathbb{R}^d$. Then for all $\beta, \gamma, \delta > 0$ we have:*

$$\|a + b\|_2^2 \leq (1 + \beta)\|a\|_2^2 + (1 + \beta^{-1})\|b\|_2^2$$
$$\|a + b + c\|_2^2 \leq (1 + \beta + \gamma)\|a\|_2^2 + (1 + \beta^{-1} + \delta)\|b\|_2^2 + (1 + \gamma^{-1} + \delta^{-1})\|c\|_2^2$$

Proof. We prove the first statement. The second follows from a similar argument. We have for $x, y \in \mathbb{R}$:

$$\beta x^2 - 2xy + \beta^{-1}y^2 = \left(\sqrt{\beta}x - \sqrt{\beta^{-1}}y\right)^2 \geq 0 \Rightarrow 2xy \leq \beta x^2 + \beta^{-1}y^2$$
$$\beta x^2 + 2xy + \beta^{-1}y^2 = \left(\sqrt{\beta}x + \sqrt{\beta^{-1}}y\right)^2 \geq 0 \Rightarrow -2xy \leq \beta x^2 + \beta^{-1}y^2$$

Therefore:

$$2|xy| \leq \beta x^2 + \beta^{-1}y^2$$

Now by Cauchy-Schwarz inequality:

$$\begin{aligned}
\|a + b\|_2^2 &= \|a\|_2^2 + 2\langle a, b \rangle + \|b\|_2^2 \\
&\leq \|a\|_2^2 + 2|\langle a, b \rangle| + \|b\|_2^2 \\
&\leq \|a\|_2^2 + 2\|a\|_2\|b\|_2 + \|b\|_2^2 \\
&\leq (1 + \beta)\|a\|_2^2 + (1 + \beta^{-1})\|b\|_2^2
\end{aligned}$$

$\square$

B. Missing proofs

B.1. Proof of Lemma 3

Proof. Taking expectation with respect to (i_k, b_k) conditional on $(i_t, b_t)_{t=1}^{k-1}$ we have:

$$
\mathbb{E}\left[\sum_{i=1}^n \left\|g_{k+1}^i - \nabla f_i(x^*)\right\|_2^2\right]
$$

$$
= \sum_{j=1}^n \mathbb{P}(i_k = j)\left[\mathbb{P}(b_k = 0 \mid i_k = j)\left(\sum_{i=1}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2\right)\right.
$$

$$
\left. + \mathbb{P}(b_k = 1 \mid i_k = j)\left(\sum_{\substack{i=1 \\ i \neq j}}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2 + \left\|\nabla f_j(x_k) - \nabla f_j(x^*)\right\|_2^2\right)\right]
$$

$$
= \sum_{j=1}^n p_k^j\left[\left(1 - \frac{\varepsilon_k}{p_k^j}\right)\sum_{i=1}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2 + \frac{\varepsilon_k}{p_k^j}\left(\sum_{\substack{i=1 \\ i \neq j}}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2 + \left\|\nabla f_j(x_k) - \nabla f_j(x^*)\right\|_2^2\right)\right]
$$

$$
= (1 - \varepsilon_k)\sum_{i=1}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2 + \varepsilon_k \sum_{i=1}^n \left\|\nabla f_i(x_k) - \nabla f_i(x^*)\right\|_2^2
$$

$$
\leq (1 - \varepsilon_k)\sum_{i=1}^n \left\|g_k^i - \nabla f_i(x^*)\right\|_2^2 + 2Ln\varepsilon_k\left[F(x_k) - F(x^*)\right]
$$

where the second and third lines follow from the update of Algorithm 1, and the last line follows from Lemma 5. □

B.2. Proof of Lemma 4

Proof. Taking expectation with respect to (i_k, b_k) conditional on $(i_t, b_t)_{t=1}^{k-1}$ we have:

$$
\mathbb{E}\left[\left\|\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right\|_2^2\right] = \frac{1}{n^2}\sum_{i=1}^n \frac{1}{p_k^i}\left\|\nabla f_i(x_k)\right\|_2^2
$$

Now by Proposition 3:

$$
\sum_{i=1}^n \frac{1}{p_k^i}\left\|\nabla f_i(x_k)\right\|_2^2 = \sum_{i=1}^n \frac{1}{p_k^i}\left\|\nabla f_i(x_k) - \nabla f_i(x^*) + \nabla f_i(x^*) - g_k^i + g_k^i\right\|_2^2
$$

$$
\leq (1 + \beta + \gamma)\sum_{i=1}^n \frac{1}{p_k^i}\left\|\nabla f_i(x_k) - \nabla f_i(x^*)\right\|_2^2
$$

$$
+ (1 + \beta^{-1} + \delta)\sum_{i=1}^n \frac{1}{p_k^i}\left\|g_k^i - \nabla f_i(x^*)\right\|_2^2
$$

$$
+ (1 + \gamma^{-1} + \delta^{-1})\sum_{i=1}^n \frac{1}{p_k^i}\left\|g_k^i\right\|_2^2
$$

Let us bound each of the three terms. The first can be bound using $p_k \geq \varepsilon_k$ and Lemma 5:

$$
\sum_{i=1}^n \frac{1}{p_k^i}\left\|\nabla f_i(x_k) - \nabla f_i(x^*)\right\|_2^2 \leq \frac{1}{\varepsilon_k}\sum_{i=1}^n \left\|\nabla f_i(x_k) - \nabla f_i(x^*)\right\|_2^2 \leq \frac{2Ln}{\varepsilon_k}\left[F(x_k) - F(x^*)\right]
$$

The second is easily bound using $p_k \geq \varepsilon_k$:

$$\sum_{i=1}^{n} \frac{1}{p_k^i} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 \leq \frac{1}{\varepsilon_k} \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2$$

Finally, the third term can be bound as:

$$\sum_{i=1}^{n} \frac{1}{p_k^i} \left\| g_k^i \right\|_2^2$$

$$\leq (1 + 2n\varepsilon_k) \left(\sum_{i=1}^{n} \left\| g_k^i \right\|_2 \right)^2$$

$$\leq (1 + 2n\varepsilon_k) \left(\sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2 + \sum_{i=1}^{n} \left\| \nabla f_i(x^*) \right\|_2 \right)^2$$

$$\leq (1 + 2n\varepsilon_k)(1 + \eta) \left(\sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x_k) \right\|_2 \right)^2 + (1 + 2n\varepsilon_k)(1 + \eta^{-1}) \left(\sum_{i=1}^{n} \left\| \nabla f_i(x^*) \right\|_2 \right)^2$$

$$\leq (1 + 2n\varepsilon_k)(1 + \eta)n \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + (1 + 2n\varepsilon_k)(1 + \eta^{-1})n^2\sigma_*^2$$

$$\leq 2n(1 + \eta) \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + (1 + 2n\varepsilon_k)(1 + \eta^{-1})n^2\sigma_*^2$$

$$\leq (1 + \eta)\frac{1}{\varepsilon_k} \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + (1 + \eta^{-1})(1 + 2n\varepsilon_k)n^2\sigma_*^2$$

Where the second line follows from Proposition 2, the third from the triangle inequality, and the fourth by Proposition 3. The fifth line follows from the definition of σ_*^2 in section 4 and an application of Cauchy-Schwarz inequality. Let v be the vector whose i^{th} component is $\left\| g_k^i - \nabla f_i(x^*) \right\|_2$ and let 1_n be the vector of ones. Then:

$$\left(\sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2 \right)^2 = |\langle 1_n, v \rangle|^2 \leq \|1_n\|_2^2 \|v\|_2^2 = n \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2$$

Finally, lines six and seven follow from the inequality $\varepsilon_k \leq 1/2n$. Combining the three bounds yields the result. $\square$

B.3. Proof of Theorem 1

Proof. Recall that we are studying the one-step evolution of the Lyapunov function:

$$T^k = T(x_k, (g_k^i)_{i=1}^n) := \frac{\alpha_k}{n\varepsilon_k} \frac{a}{L} \sum_{i=1}^{n} \left\| g_k^i - \nabla f_i(x^*) \right\|_2^2 + \|x_k - x^*\|_2^2$$

and that we are assuming (i) α_k/ε_k is constant. (ii) $\varepsilon_k \in (0, 1/2n]$. (iii) $\alpha_k \leq n\varepsilon_k/20L$.

All the expectations in this proof are with respect to (i_k, b_k) and are conditional on $(i_t, b_t)_{t=0}^{k-1}$. Since α_k/ε_k is constant, Lemma 3 immediately gives us a bound on the first term of $\mathbb{E}\left[T^{k+1}\right]$.

The second term of $\mathbb{E}\left[T^{k+1}\right]$ expands as:

$$\mathbb{E}\left[\|x_{k+1}-x^*\|_2^2\right] = \mathbb{E}\left[\left\|x_k-\alpha_k\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)-x^*\right\|_2^2\right]$$

$$= \|x_k-x^*\|_2^2 - 2\alpha_k\langle\mathbb{E}\left[\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right], x_k-x^*\rangle + \alpha_k^2\mathbb{E}\left[\left\|\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right\|_2^2\right]$$

$$= \|x_k-x^*\|_2^2 - 2\alpha_k\langle\nabla F(x_k), x_k-x^*\rangle + \alpha_k^2\mathbb{E}\left[\left\|\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right\|_2^2\right]$$

$$\leq (1-\alpha_k\mu)\|x_k-x^*\|_2 - 2\alpha_k\left[F(x_k)-F(x^*)\right] + \alpha_k^2\mathbb{E}\left[\left\|\frac{1}{np_k^{i_k}}\nabla f_{i_k}(x_k)\right\|_2^2\right]$$

where in the last line we use Assumption 1 (strong-convexity of F). Since we are assuming $\varepsilon_k\in(0,1/2n]$, we can apply Lemma 4 to bound the last term above. Combining the resulting bound with the one on the first term we get:

$$\mathbb{E}\left[T^{k+1}\right] \leq \left(1-\varepsilon_k+\frac{D_1\alpha_kL}{na}\right)\frac{\alpha_k}{n\varepsilon_k}\frac{a}{L}\sum_{i=1}^n\left\|g_k^i-\nabla f_i(x^*)\right\|_2^2 + (1-\alpha_k\mu)\|x_k-x^*\|_2^2$$

$$+ D_3(1+2n\varepsilon_k)\alpha_k^2\sigma_*^2$$

$$+ \frac{2\alpha_kL}{n\varepsilon_k}\left(D_2\alpha_k+\frac{an\varepsilon_k}{L}-\frac{n\varepsilon_k}{L}\right)\left[F(x_k)-F(x^*)\right]$$

To ensure that the last parenthesis is not positive, we need:

$$\alpha_k \leq \frac{(1-a)}{D_2}\frac{n\varepsilon_k}{L} \tag{11}$$

Assuming α_k satisfies this condition, and replacing in the first parenthesis we get:

$$1-\varepsilon_k+\frac{D_1\alpha_kL}{na} \leq 1-\alpha_k\left(\frac{D_2}{1-a}-\frac{D_1}{a}\right)\frac{L}{n}$$

Now:

$$\min\left\{\left(\frac{D_2}{1-a}-\frac{D_1}{a}\right)\frac{L}{n},\mu\right\} \leq \min\left\{\left(\frac{D_2}{1-a}-\frac{D_1}{a}\right),1\right\}\zeta$$

where:

$$\zeta := \min\left\{\frac{L}{n},\mu\right\}$$

So that, assuming (11) holds, our upper bound on $\mathbb{E}\left[T^{k+1}\right]$ is itself upper bounded by:

$$\mathbb{E}\left[T^{k+1}\right] \leq \left(1-\alpha_k\min\left\{\left(\frac{D_2}{1-a}-\frac{D_1}{a}\right),1\right\}\zeta\right)T^k + D_3(1+2n\varepsilon_k)\alpha_k^2\sigma_*^2$$

It remains to choose the parameters $\beta,\gamma,\delta,\eta>0$, and the parameter $a>0$ so as to minimize this upper bound while maximizing the step size in (11). First however, note that we have the constraint $a<1$ so that the step size can be allowed to be positive in (11). Furthermore, we choose to impose $D_3=2$ since this is also the leading factor in the analogous term in the analysis of SGD (Gower et al., 2019).

These considerations lead us to the following constrained optimization problem:

$$\max_{a,\beta,\gamma,\delta,\eta} \quad \frac{(1-a)}{D_2}$$

$$\text{subject to:} \quad \frac{D_2}{1-a} - \frac{D_1}{a} \geq 1$$

$$D_3 \leq 2$$

$$0 < a < 1$$

$$\beta, \gamma, \delta, \eta > 0$$

which we solve numerically to find the feasible point:

$$a = 0.673, \; \beta = 1.028, \; \gamma = 4.084, \; \delta = 3.973, \; \eta = 2.980$$

With this choice of parameters, we get:

$$\mathbb{E}\left[T^{k+1}\right] \leq (1 - \alpha_k \zeta) T^k + (1 + 2n\varepsilon_k)2\alpha_k^2 \sigma_*^2$$

under the condition:

$$\alpha_k \leq \frac{1}{20} \frac{n\varepsilon_k}{L}$$

which ensures that (11) holds. □

B.4. Proof of Corollary 1

Proof. Under the Assumptions of the corollary, the conditions of Theorem 1 are satisfied for all $k \in \mathbb{N}$. Starting from $\mathbb{E}\left[T^k\right]$ and repeatedly applying Theorem 1 we get:

$$\mathbb{E}\left[T^k\right] \leq (1 - \alpha\zeta)^k T^0 + (1 + 2n\varepsilon)2\alpha^2\sigma_*^2 \sum_{t=0}^{k-1}(1 - \alpha\zeta)^t$$

$$\leq (1 - \alpha\zeta)^k T^0 + (1 + 2n\varepsilon)2\alpha^2\sigma_*^2 \sum_{t=0}^{\infty}(1 - \alpha\zeta)^t$$

$$= (1 - \alpha\rho)^k T^0 + (1 + 2n\varepsilon)\frac{2\alpha\sigma_*^2}{\zeta}$$

□

B.5. Proof of Corollary 2

Proof. The choices of ε_k, α_k, and of the constants c and k_0 are such that: (i) α_k/ε_k is constant. (ii) $\alpha_k \leq n\varepsilon_k/20L$ for all $k \in \mathbb{N}$. (iii) α_k, and therefore ε_k, is decreasing for all $k \in \mathbb{N}$. (iv) $\varepsilon_0 = \frac{1}{2n}$. Therefore, the conditions of Theorem 1 hold for all $k \in \mathbb{N}$. Fix $k \in \mathbb{N}$. We have by Theorem 1 for any $t \in [k-1] \cup \{0\}$:

$$\mathbb{E}\left[T^{t+1}\right] \leq (1 - \alpha_t\zeta)\mathbb{E}\left[T^t\right] + (1 + 2n\varepsilon_t)2\alpha_t^2\sigma_*^2$$

$$= \frac{c + (t + k_0 - 1)(t + k_0 + 1)}{c + (t + k_0)(t + k_0 + 2)}\mathbb{E}\left[T^t\right] + (1 + 2n\varepsilon_t)2\alpha_t^2\sigma_*^2$$

multiplying both sides by $[c + (t + k_0)(t + k_0 + 2)]$ and noticing that:

$$[c + (t + k_0)(t + k_0 + 2)]\alpha_t^2 = \frac{[2(t + k_0) + 1]^2}{[c + (t + k_0)(t + k_0 + 2)]} \leq 4$$

we get:

$$[c + (t + k_0)(t + k_0 + 2)]\mathbb{E}\left[T^{t+1}\right] - [c + (t + k_0 - 1)(t + k_0 + 1)]\mathbb{E}\left[T^t\right] \leq (1 + 2n\varepsilon_t)8\sigma_*^2$$

summing the k inequalities for $t \in [k-1] \cup \{0\}$ and noticing that the left side is a telescoping sum we obtain:

$$[c + (k+k_0)(k+k_0+2)] \, \mathbb{E}\left[T^k\right] - [c + (k_0-1)(k_0+1)] \, T^0 \leq 8k\sigma_*^2 + 16n\sigma_*^2 \sum_{t=0}^{k-1} \varepsilon_t$$

Let us now bound the last term:

$$16n\sigma_*^2 \sum_{t=0}^{k-1} \varepsilon_t = 320L\sigma_*^2 \sum_{t=0}^{k-1} \alpha_t$$

$$= \frac{320L\sigma_*^2}{\zeta} \sum_{t=0}^{k-1} \frac{2(t+k_0)+1}{[c+(t+k_0)(t+k_0+2)]}$$

$$\leq \frac{320L\sigma_*^2}{\zeta} \int_{-1}^{k-1} \frac{2(x+k_0)+1}{[c+(x+k_0)(x+k_0+2)]} \, dx$$

$$\leq \frac{320L\sigma_*^2}{\zeta} \int_{-1}^{k-1} \frac{2(x+k_0+1)}{[c+(x+k_0)(x+k_0+2)]} \, dx$$

$$= \frac{320L\sigma_*^2}{\zeta} \log \frac{[c+(k+k_0-1)(k+k_0+1)]}{[c+(k_0-1)(k_0+1)]}$$

where the first line follows from the definition of ε_t, and the third line follows from the fact that the integrand is decreasing over the domain of integration. Using this bound and rearranging:

$$\mathbb{E}\left[T^k\right] \leq \frac{[c+(k_0-1)(k_0+1)]}{[c+(k+k_0)(k+k_0+2)]} T^0 + \frac{8k\sigma_*^2}{[c+(k+k_0)(k+k_0+2)]}$$

$$+ \frac{320L\sigma_*^2}{\zeta} \frac{1}{[c+(k+k_0)(k+k_0+2)]} \log \frac{[c+(k+k_0-1)(k+k_0+1)]}{[c+(k_0-1)(k_0+1)]}$$

$$= O\left(\frac{T^0 + \sigma_*^2 \log k}{k^2}\right) + O\left(\frac{\sigma_*^2}{k}\right)$$

$\square$

C. Details of experiments

In this section we give more details on the experiments presented in section 5.

Batch size: Our analysis is for the case where the batch size is equal to 1. We would have therefore ideally liked to run the experiments with a unit batch size. This is however extremely slow, so we used instead a batch size of $m = 128$ sampled without replacement.

Step sizes: To make a fair comparison between the algorithms, we use the same step size sequence across them for a given dataset. Define:

$$\mathcal{L} = \frac{n-m}{m(n-1)} L_{max} + \frac{n(m-1)}{m(n-1)} L$$

where $L_{max} = \max_{i \in [n]} L_i$ where L_i is the smoothness constant of f_i, L is the smoothness constant of F, and m is the batch size.

For the constant step size experiments, we use SGD as a reference, and use the maximum step size allowable in its analysis in (Gower et al., 2019), which is given by:

$$\alpha = \frac{1}{2\mathcal{L}}$$

Similarly, for the decreasing step size experiments, we use the step sizes:

$$\alpha_k = \frac{2(k+k_0)+1}{[c+(k+k_0)(k+k_0+2)]\,\mu}$$

where:

$$k_0 = \frac{4\mathcal{L}}{\mu} - 2$$
$$c = \frac{2\mathcal{L}}{\mu}$$

Algorithm specific parameters: For SRG, we initialize $g_0^i = 0$ for all $i \in [n]$, and we always update $g_k^{i_k}$, i.e. we do not sample a Bernoulli to decide whether the update occurs or not, as we believe that the added Bernoulli is only an artifact of the analysis. Furthermore, as the indices are sampled without replacement, the naive importance sampling estimator is biased. We use the mini-batch estimator introduced in (El Hanchi & Stephens, 2020) which corrects for this bias. For the constant step size experiments, we set $\varepsilon_k = \varepsilon = 1/2n$, while for the ones using decreasing step sizes, we set $\varepsilon_k = \mathcal{L}\alpha_k/n$

For SVRG, we use its loopless version (Hofmann et al., 2015) to avoid the stairlike behavior of the convergence plot of SVRG. We use a full update probability of $q = m/n$ so that the expected number of gradient evaluations per iteration is three times that of SGD, the same ratio that we would expect from running standard SVRG and SGD with a unit batch size.

For SGD with random shuffling, we sample a fresh permutation at the beginning of each epoch.

Figure: Figure 1 shows the evolution of the log relative error:

$$\log_{10}\left(\frac{\|x_k - x^*\|_2^2}{\|x_0 - x^*\|_2^2}\right)$$

For each algorithm and each dataset, we run the algorithm ten times and plot the average of the results.

D. Synthetic experiment

In this section, we compare SRG and SGD on a synthetic dataset. Our theoretical analysis suggests that the improvement of SRG over SGD is proportional to the ratio $r = \sigma^2/\sigma_*^2$. Here we construct an artificial example in which this ratio is large ($r \approx 50$), compared to $r \in [5, 10]$ for the experiments in section 5.

We do this by generating a synthetic dataset as follows:

- generate a 1000×10 features matrix A randomly with $a_{ij} \sim \mathcal{N}(0,1)$.

- generate a weight vector w randomly with $w_i \sim \mathcal{N}(0,1)$.

- generate the target values randomly as:
$$y_i = w^T a_i + \varepsilon_i$$
where each ε_i is an independent standard Cauchy random variable.

We then fit a linear regression model with the mean squared error loss using SGD and SRG. We use a batch size of 1, and a constant step size $\alpha = 1/2L$ for both. For SRG we use $\varepsilon_k = \varepsilon = 1/2n$, and initialize $g_0^i = 0$. The results are displayed in Figure 2. We ran each algorithm a hundred times and plotted the average of the results. The left plot shows the evolution of the log relative error, while the right one shows the evolution of the log of the variance of the gradient estimator $\sigma^2(x_k, 1/n)$ and $\sigma^2(x_k, p_k)$ respectively for SGD and SRG. We see that SRG reaches solution that are approximately two orders of magnitude more accurate than SGD, which is what we expect from the value of the ratio $r \approx 50$. As pointed out in section 5, it is not clear how often such large values for the ratio r are encountered in practice, but our experiments confirm that substantial gains can be achieved if it is large enough.
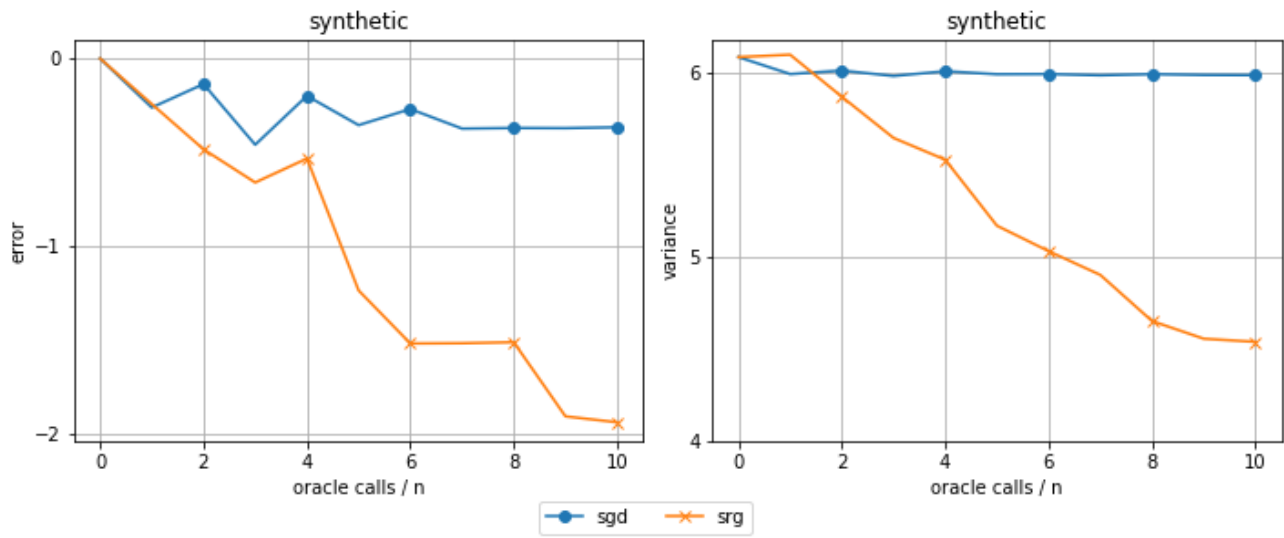
Figure 2. Comparison of the evolution of the relative error (left) and the variance of the gradient estimator (right) for SGD and SRG applied to the synthetic dataset described in section D.