An Evolving Gradient Resampling Method for Stochastic Optimization

Richard H. Byrd [*] Jorge Nocedal [†] Figen Oztoprak [‡] Stefan Solntsev [§]

January 26, 2016

Abstract

We propose an algorithm for minimizing expected risk F that shares some properties with randomized incremental aggregated gradient methods as well as dynamic sampling methods. Unlike aggregated gradient methods, which are designed to revisit the training set many times, the algorithm proposed here is well suited for problems involving a very large training set, where one (or a few) passes over the data suffice to produce an acceptable solution. At every iteration, the algorithm updates a collection of gradients from certain past iterations, and as in dynamic sample methods *additional* gradients are evaluated at the current point. By allowing the amount of information to increase at every iteration the algorithm is able to achieve linear convergence in expected risk F (not just in training error). Numerical results on machine learning test problems illustrate the performance of the method.

1 Introduction

The stochastic gradient method of Robbins and Monro [13] is the algorithm of choice for optimization problems arising in many large-scale machine learning applications. At the onset of the optimization procedure, when the number of processed datapoints is relatively small, it often outperforms most other methods proposed in the literature. However, to achieve such efficient initial behavior, the steplength must be selected in a way that prevents fast progress in later iterations. This is due to the high variance of the stochastic gradients, which must be controlled with a diminishing steplength. Several algorithms have been proposed to address this limitation, including methods that average the iterates [12, 15, 10], dynamic sampling methods [3, 6, 11] and aggregated gradient methods [14, 8, 16, 9, 5, 7, 4]. The latter focus on the finite sum problem and enjoy a linear rate of convergence for strongly convex problems, whereas the stochastic gradient method (in this case, the incremental gradient method) only has a sublinear rate of convergence.

[*] Department of Computer Science, University of Colorado, Boulder, CO, USA. This author was supported by National Science Foundation grant DMS-1216554 and Department of Energy grant DE-SC0001774.

[†] Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA. This author was supported by National Science Foundation grant DMS-0810213, and by Department of Energy grant DE-FG02-87ER25047.

[‡] Istanbul Bilgi University. This author was supported by Department of Energy grant DE-SC0001774, and by a grant from Tubitak.

[§] Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA. This author was supported by National Science Foundation grant DMS-0810213.

In this paper, we propose an algorithm that shares some characteristics with the aggregated gradient methods as well as with dynamic sampling methods (sometimes called batching methods). Unlike aggregated gradient methods, which are designed to revisit the training set multiple times (perhaps dozens of times), the algorithm proposed here is well suited for problems involving an extremely large training set, and where one (or a few) passes over the data suffice to produce an acceptable solution. The method is quite general; it includes as special cases the classical stochastic gradient method, the aggregated gradient methods SAG [4] and SAGA [4] and dynamic sampling methods [3, 6, 11].

The goal of this work is the design of an algorithm that achieves good testing error — not just good training error — a goal it shares with the methods recently proposed in [7, 1]. The novelty of our analysis is that it describes the behavior of a method that updates old gradients *and also* increases the batch size at each iteration. This is of interest in its own right due to the generality of the framework, and is also of practical value as an algorithm that strikes the right balance between these two noise reducing techniques could prove to be effective in machine learning applications.

The problem of interest is stated as

$$\min_{x \in \mathbb{R}^n} F(x) = \mathbb{E}[f(x;\xi)], \tag{1.1}$$ {risk}

where ξ is a random variable with distribution P and $f(\cdot\,;\xi) : \mathbb{R}^n \to \mathbb{R}$ is a smooth function. In machine learning applications, f is the composition of a loss function and a prediction function parametrized by a vector x. For purposes of analysis, we assume that $f(\cdot\,;\xi)$ is strongly convex, but the algorithm is well-defined and applicable to non-convex problems.

The stochastic gradient (SG) method for solving problem (1.1) is given by

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k;\xi_k), \tag{1.2}$$ {sgdm}

where α_k is a steplength and ξ_k is a realization of the random variable ξ. The method proposed in this paper has the more general form

$$x_{k+1} = x_k - \alpha y_k, \tag{1.3}$$ {iteration}

where y_k is computed as a weighted average of the sample gradients $\nabla f(x_i, \xi_j)$ evaluated at previous iterations and sample gradients $\nabla f(x_k, \xi_j)$ evaluated at the current iterate x_k. (Here ξ_j simply stands for some past realizations of the random variable ξ.) The algorithm is designed so that the amount of gradient information contained in y_k increases monotonically during the course of the optimization procedure. This improves the quality of the vector y_k compared to the aggregated gradient methods mentioned above and allows the algorithm to achieve a low value of the objective F. We refer to the algorithm as the Evolving Gradient Resampling Method, or EGR in short. We show that, for strongly convex objective functions, the expected value of $F(x_k)$ generated by one instance of the EGR method converges to the optimal value F^* at a linear rate.

The paper is organized into 6 sections. In the next section we present the algorithm, and in section 3 we study its convergence properties. A detailed discussion of the implementation of the algorithm is given in section 4 and the results of numerical experiments are reported in section 5. The contributions of the work and its relationship to other works in the literature are summarized in section 6.

Notation and Definitions. In our analysis we assume that F is strongly convex and denote its minimizer by x^*, and the corresponding objective value by F^*. Following Bertsekas [2] we use the term “incremental gradient method” for the method designed to minimize a finite sum problem and “stochastic gradient” for the same method for minimizing empirical risk F.

2 The Evolving Gradient Resampling (EGR) Method

{themethod}

The EGR method combines ideas from dynamic sampling methods, which employ batches of increasing size, and aggregated gradient methods that store gradients evaluated at previous iterations. The latter, which are more accurately described as *randomized incremental aggregated methods* [14, 8, 7, 4], employ an iteration of the form (1.3) to minimize the finite sum

$$\psi(x) = \frac{1}{m}\sum_{i=1}^{m} f^i(x), \tag{2.1}$$ {saa}

where $f^i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ are given convex functions. An example of a method of this kind is SAG [14], where at every iteration k an index $j \in \{1, \ldots, m\}$ is chosen at random and the aggregated gradient y_k is computed as

$$y_k^{sag} = \frac{1}{m}\left[\nabla f^j(x_k) - \nabla f^j(\phi_{k-1}^j) + \sum_{i=1}^{m} \nabla f^i(\phi_{k-1}^i)\right]. \tag{2.2}$$ {sag}

Here ϕ_{k-1}^i is the most recent iterate at which the gradient of the function f^i was evaluated. Thus, in the SAG method one gradient, namely ∇f^j, is updated while the rest of the gradients kept in storage (the summation in (2.2)) are not altered. A distinctive property of SAG and the methods described in [8, 4, 16, 9, 5] is that they achieve a linear rate of convergence in $\mathbb{E}[\psi(x_k)]$, which is not the case for the incremental gradient method. Numerical experience has shown that the SAG, SAGA [4] and SVRG [8] methods are effective in practice.

Aggregated gradient methods are designed for the case when m is not too large so that multiple passes over the data are possible. Storage is a concern, as m gradients need to be saved, which may not be practical in general for applications involving large data sets. However, when f^i is the composition of a linear function and a smooth function (as in logistic regression) it suffices to store one scalar in lieu of the gradient vector [14]. Storage is not an issue for the SVGR method [8], which requires only one extra vector of storage regardless of the form of f^i.

The EGR method maintains an integer t_k that counts the number of gradient vectors in storage at the beginning of the k-th iteration, and defines the set

$$T_k = \{1, \ldots, t_k\}. \tag{2.3}$$ {capt}

Thus, T_k is the set of the assigned indices to the sampled gradients up to iteration k; i.e. $i \in T_k$ is the index corresponding to the i^{th} realization of ξ.

The objective function F can then be approximated by the sample average approximation

$$F_{T_k}(x) = \frac{1}{t_k}\sum_{i=1}^{t_k} f(x;\xi_i) \;\equiv\; \frac{1}{t_k}\sum_{i=1}^{t_k} f^i(x), \tag{2.4}$$ {batch}

where we have defined

$$f^i(x) = f(x, \xi_i), \tag{2.5}$$ {spring}

so that F_{T_k} is similar in form to the (deterministic) finite sum function (2.1). The iteration of the EGR method is given by

$$x_{k+1} = x_k - \alpha y_k, \tag{2.6}$$ {egri}

where α is a (fixed) steplength and the aggregated gradient y_k is defined as follows. To compute the k-th step, the method first adds some new sample gradients, indexed by the set U_k, to the current aggregated gradient, and at the same time updates some of the gradients previously stored, indexed by S_k. More precisely, defining

$$u_k = |U_k|, \qquad s_k = |S_k|, \tag{2.7}$$ {usdef}

the first version of our method sets

$$y_k = \frac{1}{t_k + u_k} \left(\sum_{j \in S_k} \left[\nabla f^j(x_k) - \nabla f^j(\phi^j_{k-1}) \right] + \sum_{i=1}^{t_k} \nabla f^i(\phi^i_{k-1}) + \sum_{j \in U_k} \nabla f^j(x_k) \right), \tag{2.8}$$ {ysag}

where ϕ^j_{k-1} denotes the most recent iterate at which the gradient of function the f^j was computed (we define this precisely below).

Formula (2.8) is motivated by the SAG method (2.2). The main changes are that the second sum in (2.8) includes t_k terms rather than m terms, that the first sum accounts for the fact that s_k gradients in storage are updated, and the last sum represents the addition of u_k gradients evaluated at the current iterate (i.e., a mini-batch computation). Thus, in the EGR method we have

$$t_{k+1} = t_k + u_k, \tag{2.9}$$ {tdef}

and by choosing $u_k > 0$ the number of gradients contained in y_k grows at every iteration in contrast with methods described in [8, 4, 16, 9, 5], where the number of stored gradients is constant. (The SAG method corresponds to the choice $s_k = 1, u_k = 0, t_k = m$).

Another version of the EGR method is based on the SAGA [4] step direction,

$$y_k^{saga} = \nabla f^j(x_k) - \nabla f^j(\phi^j_{k-1}) + \frac{1}{m} \sum_{i=1}^{m} \nabla f^i(\phi^i_{k-1}). \tag{2.10}$$ {saga}

Allowing for multiple sample gradient replacements as well as for the addition of new gradients to memory, the version of EGR based on SAGA defines the search direction as

$$y_k = \frac{1}{s_k + u_k} \left(\sum_{j \in S_k} \left[\nabla f^j(x_k) - \nabla f^j(\phi^j_{k-1}) + \frac{1}{t_k} \sum_{i=1}^{t_k} \nabla f^i(\phi^i_{k-1}) \right] + \sum_{j \in U_k} \nabla f^j(x_k) \right), \tag{2.11}$$ {ysaga}

where, as before, ϕ^j_{k-1} denotes the most recent iterate at which the gradient of the function f^j was computed. A pseudocode of the algorithm, which can employ either definition of y_k is given in Algorithm 1.

Algorithm 1 EGR Algorithm {alg1}

```
Input: Sequences $\{s_k\}$, $\{u_k\}$, steplength $\alpha$, and initial iterate $x_0$
Set $t_0 = 0$
loop $k = 0, 1, 2, , \dots$
    $U_k = \{t_k + 1, \dots, t_k + u_k\}$
    For all $j \in U_k$ draw $\xi_j$ from the distribution $P$
    Define $S_k$ as a set of $s_k$ indices from $T_k = \{1, \dots, t_k\}$, sampled uniformly
    Evaluate $\nabla f^j(x_k) = \nabla f(x_k, \xi_j)$ for $j \in S_k \cup U_k$
    Compute $y_k$ by (2.8) or (2.11)
    Set
        $\phi_k^i = \begin{cases} x_k & \text{for all } i \in S_k \cup U_k \\ \phi_{k-1}^i & \text{for all } i \in T_k \setminus S_k \end{cases} \qquad (2.12)$ {chico}
    $x_{k+1} = x_k - \alpha y_k$
    $t_{k+1} = t_k + u_k$
end loop
```

Thus, for component functions f^i indexed by $S_k \cup U_k$, gradients are computed at the current iterate x_k . The set U_k indexes previously unseen component functions and S_k component functions evaluated at earlier iterations whose gradients will be updated.

As stated, Algorithm 1 is very general as we have not specified the sequences $\{u_k\}$ and $\{s_k\}$ that determine the number of unseen and seen datapoints processed at iteration k. Different choices for the sequences $\{u_k\}$ and $\{s_k\}$ give rise to various algorithms, some known and some new. For example, the stochastic gradient method (1.2) is obtained by setting $s_k = 0, u_k = 1$ for all k, and dynamic sampling methods are obtained by setting $s_k = 0$ and letting u_k be an increasing sequence (both using update formula (2.11)). The purpose of allowing additional samples ($u_k > 0$) at every iteration is to achieve an increasingly accurate approximation y_k of the gradient $\nabla F(x_k)$.

Various other examples of $\{u_k\}$ and $\{s_k\}$ are discussed in section 4, including the two natural choices $u_k = O(k)$ and $u_k = O(a^k)$ for some $a > 1$. The rest of the paper is devoted to an investigation of the properties the EGR method. In the next section we present a convergence analysis for the SAG version of EGR in the case when the sequences $\{u_k\}$ and $\{s_k\}$ grow geometrically, and in section 5 we report numerical results for both versions of the EGR method using several choices for these sequences.

It is pertinent at this point to mention some recently proposed aggregated gradient methods that, like EGR, aim to yield a linear rate of convergence in testing error F, not just in training error ψ. The streaming SVRG method described by Frostig et al [7] computes a mini-batch gradient of random size (as opposed to size m) at the beginning of every cycle of SVRG, whereas the method proposed in [1] gradually grows the size of this batch gradient, thus having the flavor of a dynamic sampling method. A linear rate of convergence in the expected value of F is established in these papers.

An algorithm that is more closely related to EGR is the *initialization heuristic* of SAG. This technique gradually grows the number of stored gradients in SAG until the full storage of m vectors is reached. At each iteration of this method, a randomly sampled integer $j \in \{1, ..., m\}$ is chosen and the corresponding sample gradient $\nabla f^j(x_k)$ is evaluated at the current iterate. If j had been sampled previously this new gradient replaces the old one. Otherwise the new gradient is stored, and the memory grows; see [14] for details. The main difference between this method and EGR is that in our framework we pre-determine the sequences u_k and s_k that specify the number of new and updated gradients, whereas in the initialization heuristic of SAG, the choice of sampling from S_k or U_k is random. One can employ a similar initialization technique for SAGA, and both versions are tested in Section 5.

3 Convergence Analysis

{analysis}

We now study the convergence properties of an instance of Algorithm 1 in which both the number of new samples u_k and the number of recomputed gradients s_k grow geometrically, and y_k is given by (2.8). At the beginning of the k-th iteration, the algorithm has t_k gradients in storage indexed by the set $T_k = \{1, \dots, t_k\}$, constructs the sets

$$S_k \subseteq T_k, \quad U_k \not\subseteq T_k, \qquad \text{and defines the new set} \quad T_{k+1} = T_k \cup U_k. \tag{3.1}$$

{capusu}

We make the following assumptions about the construction of the sets S_k and U_k, and about the objective function F.

Assumptions

A.1. Let $s_k = u_k = \eta(1+\eta)^{k-1}$, with $\eta \in (0,1)$, and $u_0 = 1$. We don't get integers [Figen: If the value $\eta(1+\eta)^{k-1}$ is rounded up or down, I think we cannot get (3.9). Why not directly requiring that $u_k \geq \eta t_k$, and that $i \in T_k$ be included in S_k with probability η–so that $E[s_k] = \eta t_k$?] This implies (see derivation below) that η is the probability that an index $i \in T_k = \{1, \dots, t_k\}$ is included in S_k.

A.2. Each sample function f^i (see (2.5)) has a Lipschitz continuous gradient with parameter L_i, and the objective function F given in (1.1) has a Lipschitz continuous gradient with parameter L_F. This enables us to define a bound $L < \infty$ on the average of constants L_i:

$$L = \max \left\{ L_F, \; \max_k \left\{ \frac{1}{t_k} \sum_{i=1}^{t_k} L_i \right\} \right\}. \tag{3.2}$$

{eq:L}

[JN: should the max by sup?][Figen: $k \to \infty$, I think we can prefer sup]

A.3. F is strongly convex with a strong convexity parameter μ.

A.4. The trace of the variance of individual sample gradients is uniformly bounded. That is, there exists a positive scalar v such that

$$\operatorname{tr}(\operatorname{Var}(\nabla f^i(x))) \leq v^2, \; \forall x \in \mathbb{R}^n.$$

3.1 Some Definitions

In order to analyze the convergence properties of the algorithm, we define additional quantities.

We first consider the distance between the direction y_k and the gradient $\nabla F_{T_{k+1}}(x)$ of the sample approximation (2.4). [JN: this is where the fact that the indices are shifted makes the notation confusing][Figen: We can define another set $I_k = T_k \cup U_k (= T_{k+1})$ not to have the iteration counter shifted]. For this purpose, we define the individual errors for all samples observed up to, and including, iteration k:

$$e_k^i = \nabla f^i(\phi_k^i) - \nabla f^i(x_k), \quad i \in T_k \cup U_k. \tag{3.3}$$ {eri}

It follows from (2.12) that

$$e_k^i = 0 \;\text{ for } i \in S_k \cup U_k. \tag{3.4}$$ {zeros}

Recalling (2.9), the first statement in (2.3) and (2.12), we see that (2.8) can be written as

$$\begin{aligned}
y_k &= \frac{1}{t_{k+1}} \left(\sum_{i \in S_k} \nabla f^i(x_k) + \sum_{i \in (T_k \setminus S_k)} \nabla f^i(\phi_{k-1}^i) + \sum_{i \in U_k} \nabla f^i(x_k) \right) &(3.5)\\
&= \frac{1}{t_{k+1}} \left(\sum_{i \in S_k} \nabla f^i(x_k) + \sum_{i \in (T_k \setminus S_k)} \nabla f^i(\phi_k^i) + \sum_{i \in U_k} \nabla f^i(x_k) \right) &(3.6)\\
&= \frac{1}{t_{k+1}} \left(\sum_{i \in U_k \cup T_k} \nabla f^i(x_k) + \sum_{i \in (T_k \setminus S_k)} [\nabla f^i(\phi_k^i) - \nabla f^i(x_k)] \right). &(3.7)
\end{aligned}$$ {gk_sag2}

Therefore, from (2.4), (2.9) and (3.4)

$$y_k = \nabla F_{T_{k+1}}(x_k) + \frac{1}{t_{k+1}} \sum_{i=1}^{t_k} e_k^i. \tag{3.8}$$ {eq:y}

By assumption A.1, we have that

$$t_k = \sum_{i=0}^{k-1} u_i \geq (1+\eta)^{k-1} \;\text{ and }\; t_{k+1} = \sum_{i=0}^{k} u_i = (1+\eta)^k \geq (1+\eta) t_k. \tag{3.9}$$ {kkasdf}

At each iteration k, an index $i \in T_k = \{1, \ldots, t_k\}$ is included in S_k with probability

$$\frac{s_k}{t_k} \geq \frac{\eta(1+\eta)^{k-1}}{(1+\eta)^{k-1}} = \eta. \tag{3.10}$$

So, from (2.12),

$$\phi_k^i = \begin{cases} x_k & \text{if } i \in U_k, \\ x_k & \text{with probability } \eta, \;\text{ if } i \in T_k, \\ \phi_{k-1}^i & \text{with probability } 1-\eta, \;\text{ if } i \in T_k, \end{cases}$$

and therefore, we have from (3.3) that for $i \in T_k$

$$\|e_k^i\| = \begin{cases} 0 & \text{with probability } \eta, \\ \|e_{k-1}^i + \nabla f^i(x_{k-1}) - \nabla f^i(x_k)\| & \text{with probability } 1-\eta. \end{cases}$$

Then, using Assumption A.2.,

$$\mathbf{E}_k[\|e_k^i\|] = (1-\eta)\|e_{k-1}^i + \nabla f^i(x_{k-1}) - \nabla f^i(x_k)\| \leq (1-\eta)\|e_{k-1}^i\| + (1-\eta) L_i \|x_{k-1} - x_k\|, \tag{3.11}$$ {eq:expec}

where $\mathbf{E}_k$ is the expectation over all choices of S_k at iteration k given x_{k-1} and e_{k-1}.

We also define δ_k to be the the following bound over all individual errors e_k^i:

$$\delta_k = \frac{1}{t_{k+1}} \sum_{i=1}^{t_{k+1}} \|e_k^i\| = \frac{1}{t_{k+1}} \sum_{i=1}^{t_k} \|e_k^i\|, \text{ for } k > 0, \quad \text{and } \delta_0 = 0, \tag{3.12}$$

{eq:defe}

where the first equality follows from $t_{k+1} = t_k + u_k$ and (3.4). Therefore, we have from (3.8) that

$$\|y_k\| \leq \|\nabla F_{T_{k+1}}(x_k)\| + \delta_k. \tag{3.13}$$

{yfe}

Next, we estimate the difference between the gradients of the sample approximation $F_{T_k}(x)$ and the objective function F. Let $\mathbf{E}[\cdot]$ denote the expectation with respect to all the choices of the sets U_ℓ, S_ℓ for $\ell = 1, 2 \ldots$. By Jensen's inequality,

$$\begin{aligned}
(\mathbf{E}[\|\nabla F_{T_k}(x) - \nabla F(x)\|])^2 &\leq \mathbf{E}[\|\nabla F_{T_k}(x) - \nabla F(x)\|^2] \\
&= \mathbf{E}[\mathrm{tr}(\nabla F_{T_k}(x) - \nabla F(x))(\nabla F_{T_k}(x) - \nabla F(x))^T] \\
&= \mathrm{tr}(\mathrm{Var}(\nabla F_{T_k}(x))) \\
&= \mathrm{tr}\left(\mathrm{Var}\left(\frac{1}{t_k}\sum_{i=1}^{t_k} \nabla f^i(x)\right)\right) \\
&\leq \frac{1}{t_k}\mathrm{tr}(\mathrm{Var}(\nabla f^i(x))) \\
&\leq \frac{v^2}{t_k},
\end{aligned}$$

where the last equality follows from Assumption A.4, and the fact that ξ_i are independent. [Figen: For the second line of the above derivation, is it straigtforward to see that $E[\nabla F_{T_k}(x)] = \nabla F(x)$, or should we discuss it?] If we define

$$\sigma_{k-1} = \sqrt{v^2/t_k} \tag{3.14}$$

{sigmad}

we can write

$$\mathbf{E}\|\nabla F_{T_k}(x) - \nabla F(x)\| \leq \sigma_{k-1} \qquad \text{for all } x \in \mathbb{R}^n. \tag{3.15}$$

{vat}

Now, since the number of observed data grows geometrically, i.e. $t_{k+1} \geq (1+\eta)t_k$, the bound σ_k decays geometrically,

$$\sigma_k \leq \sqrt{\frac{v^2}{(1+\eta)t_k}} = \sqrt{\frac{1}{1+\eta}}\sigma_{k-1}. \tag{3.16}$$

{sigma}

3.2 Linear Convergence of Algorithm 1

We are now ready to establish the main convergence results of the paper. The following lemma describes the joint evolution of three quantities: δ_k, σ_k and $\|x_k - x_*\|$, where x_* is the minimizer of F. (The sum of these 3 quantities can be seen as a Lyapunov function.) We recall that $\mathbf{E}[\cdot]$ denotes the expectation with respect to all the choices of the sets U_ℓ, S_ℓ, for $\ell = 1, 2, \ldots$, and that the constant L is defined in (3.2).

{lemma:main}

Lemma 1. *Let $\{x_k\}$ be the iterates produced by Algorithm 1 with $\alpha \leq \frac{1}{L}$ [Figen:Do we use the assumption on α?] and y_k given by (2.8), and suppose that Assumptions A1 - A4 hold. Then,*

$$\begin{pmatrix} \frac{1}{L}\mathbf{E}[\mathbf{E}_k[\delta_k]] \\ \mathbf{E}[\|x_k - x_*\|] \\ \sigma_k \end{pmatrix} \leq M \begin{pmatrix} \frac{1}{L}\mathbf{E}[\delta_{k-1}] \\ \mathbf{E}[\|x_{k-1} - x_*\|] \\ \sigma_{k-1} \end{pmatrix}, \tag{3.17}$$

{control}

where

$$M = \begin{pmatrix} \left(\frac{1-\eta}{1+\eta}\right)(1+\alpha L) & \left(\frac{1-\eta}{1+\eta}\right)\alpha L & \left(\frac{1-\eta}{1+\eta}\right)\alpha \\ \alpha L & 1-\alpha\mu & \alpha \\ 0 & 0 & \sqrt{\frac{1}{1+\eta}} \end{pmatrix}, \tag{3.18}$$

{eq:M}

and x_* is the unique minimizer of (1.1).

Proof. The last equation in (3.17) follows from (3.16).

Let us establish the first equation. By using (3.12), (3.11), (3.9), (2.6), (3.2), (3.8), (3.13) and (3.15), we obtain

$$\begin{aligned}
\mathbf{E}[\mathbf{E}_k[\delta_k]] =& \frac{1}{t_{k+1}}\sum_{i=1}^{t_k}\mathbf{E}[\mathbf{E}_k[\|e_k^i\|]] \\
\leq& \frac{1}{t_{k+1}}\sum_{i=1}^{t_k}\big((1-\eta)\mathbf{E}\|e_{k-1}^i\| + (1-\eta)L_i\mathbf{E}\|x_{k-1}-x_k\|\big) \\
\leq& (1-\eta)\frac{1}{t_{k+1}}\left(\sum_{i=1}^{t_k}\mathbf{E}\|e_{k-1}^i\|\right) + (1-\eta)\mathbf{E}\|\alpha y_{k-1}\|\frac{1}{t_{k+1}}\sum_{i=1}^{t_k}L_i \\
\leq& \left(\frac{1-\eta}{1+\eta}\right)\mathbf{E}\left(\frac{1}{t_k}\sum_{i=1}^{t_k}\|e_{k-1}^i\|\right) + \left(\frac{1-\eta}{1+\eta}\right)L\alpha\left(\mathbf{E}\|\nabla F_{T_k}(x_{k-1})\| + \mathbf{E}[\delta_{k-1}]\right) \\
\leq& \left(\frac{1-\eta}{1+\eta}\right)(1+L\alpha)\,\mathbf{E}[\delta_{k-1}] + \left(\frac{1-\eta}{1+\eta}\right)L\alpha\mathbf{E}\,\|\nabla F_{T_k}(x_{k-1})\| \\
=& \left(\frac{1-\eta}{1+\eta}\right)(1+L\alpha)\,\mathbf{E}[\delta_{k-1}] \\
&+ \left(\frac{1-\eta}{1+\eta}\right)L\alpha\mathbf{E}\,\|\nabla F_{T_k}(x_{k-1}) - \nabla F(x_{k-1}) + \nabla F(x_{k-1}) - \nabla F(x_*)\| \\
\leq& \left(\frac{1-\eta}{1+\eta}\right)(1+L\alpha)\,\mathbf{E}[\delta_{k-1}] \\
&+ \left(\frac{1-\eta}{1+\eta}\right)L\alpha\sigma_{k-1} + \left(\frac{1-\eta}{1+\eta}\right)L\alpha\mathbf{E}\|\nabla F(x_{k-1}) - \nabla F(x_*)\|.
\end{aligned}$$

Therefore,

$$\mathbf{E}[\mathbf{E}_k[\delta_k]] \leq \frac{1-\eta}{1+\eta}(1+L\alpha)\,\mathbf{E}[\delta_{k-1}] + \frac{1-\eta}{1+\eta}L^2\alpha\mathbf{E}\|x_{k-1}-x_*\| + \frac{1-\eta}{1+\eta}L\alpha\sigma_{k-1}, \tag{3.19}$$

{eq:row1}

which proves the first inquality in (3.17). To establish the second inequality, we note from (2.6) and

(3.13) that

$$\begin{aligned}
\mathbf{E}\|x_k - x_*\| &= \mathbf{E}\|x_{k-1} - x_* + x_k - x_{k-1}\|\\
&= \mathbf{E}\|x_{k-1} - x_* - \alpha y_{k-1}\|\\
&= \mathbf{E}\left\|x_{k-1} - x_* - \alpha\left(\nabla F_{T_k}(x_{k-1}) + \delta_{k-1} + \nabla F(x_{k-1}) - \nabla F(x_{k-1})\right)\right\|\\
&\le \mathbf{E}\|x_{k-1} - x_* - \alpha\nabla F(x_{k-1})\| + \alpha\mathbf{E}\|\nabla F_{T_k}(x_{k-1}) - \nabla F(x_{k-1})\| + \alpha\mathbf{E}[\delta_{k-1}]\\
&\le \mathbf{E}\left\|\left(I - \alpha\int_{t=0}^{1} \nabla^2 F(tx_{k-1} + (1-t)x_*)\right)(x_{k-1} - x_*)\right\| + \alpha\sigma_{k-1} + \alpha\mathbf{E}[\delta_{k-1}].
\end{aligned}$$

[JN: we have not assumed that F is twice differentiable. We either make that assumption or re-derive the last line using pure convexity] By Assumption A.3 and the assumption that $\alpha \le \frac{1}{L}$ [Figen:Do we use this assumption on α?] we have

$$\mathbf{E}\|x_k - x_*\| \le (1-\alpha\mu)\mathbf{E}\|x_{k-1} - x_*\| + \alpha\sigma_{k-1} + \alpha\mathbf{E}[\delta_{k-1}]. \tag{3.20}$$

{eq:row2}

Now, gathering (3.19), (3.20), and (3.16) we obtain (3.17). □

The next Lemma describes a bound on the spectral radius ρ_M of the matrix M defined in (3.18). This bound provides the rate of convergence of EGR as will be shown in Theorem 3, and is further elaborated upon in Remark 5.

{lemma:spect

Lemma 2. *Choose the constant steplength*

$$\alpha = \min\left\{\beta\left(\frac{2\eta}{1-\eta}\right)\frac{\mu}{L}\frac{1}{\mu+L}, \frac{1}{L}\right\}, \quad \textit{for some } \beta \in (0,1). \tag{3.21}$$

{eq:alpha}

The spectral radius $\lambda_{\max}^M$ of the matrix M in (3.18) *then satisfies*

$$\sqrt{\frac{1}{\eta+1}} \le \lambda_{\max}^M \le \max\left\{\sqrt{\frac{1}{\eta+1}}, 1 - \frac{\alpha\left(2\eta\mu + \alpha(\eta-1)L^2 + \alpha(\eta-1)\mu L\right)}{\alpha((\eta+1)\mu - (1-\eta)L) + 2\eta}\right\} < 1. \tag{3.22}$$

{eq:spectral

Proof. The 3×3 matrix (3.18) has three eigenvalues; we have $\lambda_3^M = \sqrt{\frac{1}{1+\eta}}$, and λ_1^M, λ_2^M are the eigenvalues of upper-left 2×2 block

$$B = \begin{pmatrix} \left(\frac{1-\eta}{1+\eta}\right)(1+\alpha L) & \left(\frac{1-\eta}{1+\eta}\right)\alpha L \\ \alpha L & 1-\alpha\mu \end{pmatrix}. \tag{3.23}$$

Let

$$A = \begin{pmatrix} \left(\frac{1-\eta}{1+\eta}\right)(1+\alpha L) - 1 & \left(\frac{1-\eta}{1+\eta}\right)\alpha L \\ \alpha L & -\alpha\mu \end{pmatrix}$$

so that

$$B = I + A.$$

We will first show that both eigenvalues of A, λ_1^A, λ_2^A, are negative.
Case I. In (3.21), $\alpha = \beta\left(\frac{2\eta}{1-\eta}\right)\frac{\mu}{L}\frac{1}{\mu+L}$.

$$A_{11} = \left(\frac{1-\eta}{1+\eta}\right)(1+\alpha L) - 1 = -\frac{2\eta(-\beta\mu + \mu + L)}{(\eta+1)(\mu+L)} < 0.$$

So, both $A_{11} < 0$ and $A_{22} < 0$, yielding $\mathrm{tr}(A) < 0$. On the other hand,

$$\begin{aligned}
\det(A) &= \left(\left(\frac{1-\eta}{1+\eta}\right)(1+\alpha L) - 1\right)(-\alpha\mu) - \left(\frac{1-\eta}{1+\eta}\right)\alpha^2 L^2 \\
&= \frac{\alpha\left(2\eta\mu - \alpha(1-\eta)L^2 - \alpha(1-\eta)\mu L\right)}{\eta+1} \\
&= \frac{4(1-\beta)\beta\eta^2\mu^2}{\left(1-\eta^2\right)L(\mu+L)} \\
&> 0
\end{aligned}$$

Case II. In (3.21), $\alpha = \frac{1}{L}$. Note that in this case

$$\beta\left(\frac{2\eta}{1-\eta}\right)\frac{\mu}{\mu+L} \geq 1 \quad \Leftrightarrow \quad \left(\frac{1-\eta}{1+\eta}\right) \leq \left(\frac{1+\kappa}{\beta}+1\right)^{-1} < (\kappa+2)^{-1}, \qquad (3.24) \quad \texttt{\{eq:step\}}$$

where $\kappa = \dfrac{L}{\mu}$. Then,

$$\mathrm{tr}(A) < A_{11} = \left(\frac{1-\eta}{1+\eta}\right)(2) - 1 \leq 2\left(\kappa+2\right)^{-1} - 1 < 0,$$

and

$$\begin{aligned}
\det(A) &= \left(\left(\frac{1-\eta}{1+\eta}\right)(2) - 1\right)(-\frac{1}{L}\mu) - \left(\frac{1-\eta}{1+\eta}\right) \\
&= \left(1 - \left(\frac{1-\eta}{1+\eta}\right)(2)\right)(\kappa) - \left(\frac{1-\eta}{1+\eta}\right) \\
&= \kappa + (-2\kappa - 1)\left(\frac{1-\eta}{1+\eta}\right) \\
&\geq \kappa + \frac{-2\kappa-1}{\kappa+2} \\
&> 0.
\end{aligned}$$

Therefore we conclude that $\lambda_1^A, \lambda_2^A < 0$. Let us number the eigenvalues of A so that $\lambda_1^A \geq \lambda_2^A$. To bound λ_1^A above note that

$$\frac{\det(A)}{\mathrm{tr}(A)} = \frac{\lambda_1^A \lambda_2^A}{\lambda_1^A + \lambda_2^A} > \lambda_1^A$$

since $\lambda_2^A/(\lambda_1^A + \lambda_2^A) \leq 1$. Therefore,

$$\begin{aligned}
\lambda_{\max}^A <& \frac{\det(A)}{\mathrm{tr}(A)} = -\frac{\alpha\left(2\eta\mu + \alpha(\eta-1)L^2 + \alpha(\eta-1)\mu L\right)}{(\eta+1)\left(\alpha\mu - \frac{(1-\eta)(\alpha L+1)}{\eta+1} + 1\right)} \qquad (3.25) \quad \texttt{\{eq:lambdaA\}} \\
&= -\frac{\alpha\left(2\eta\mu + \alpha(\eta-1)L^2 + \alpha(\eta-1)\mu L\right)}{\alpha((\eta+1)\mu - (1-\eta)L) + 2\eta} \\
&<0.
\end{aligned}$$

Since $\lambda^M_{\max} = \max\{\sqrt{\frac{1}{1+\eta}}, \lambda^M_1, \lambda^M_2\}$ and we have $\lambda^M_1 = 1 + \lambda^A_1$, $\lambda^M_2 = 1 + \lambda^A_2$, the upper and lower bound result follows. □

{thm:linear}

Theorem 3. *Suppose Assumptions A.1-A.4 hold, and y_k is computed with* (2.8)*. Then, $\{\mathbf{E}\|x_k - x_*\|\}$ produced by Algorithm 1 converges to zero R-linearly.*

Proof. By Lemma 1,

$$\begin{pmatrix} \frac{1}{L}\mathbf{E}[\delta_k] \\ \mathbf{E}[\|x_k - x_*\|] \\ \sigma_k \end{pmatrix} \le M^k \begin{pmatrix} \frac{1}{L}\delta_0 \\ \|x_0 - x_*\| \\ \sigma_0 \end{pmatrix}. \tag{3.26}$$

[JN: note that $\delta_0 = 0$, but I suppose that is OK] [Figen: Shouldn't we have $\mathbf{E}[\mathbf{E}^k[\delta_k]]$ on the first line? To study the norm of M we note that since the upper left 2×2 matrix is positive, it has 2 distinct eigenvalues, and thus 2 linearly independent eigenvectors. If we append a zero to these vectors we have 2 independent eigenvectors of M which are clearly independent of the eigenvector corresponding to $\sqrt{\frac{1}{1+\eta}}$. If we define S to be the matrix whose columns are these 3 eigenvectors, we have that $S^{-1}MS = D$ where D is diagonal. Now if we define the weighted vector norm

$$\|x\|_S \equiv \|S^{-1}x\|_2,$$

then the corresponding induced matrix norm of M is

$$\|M\|_S \equiv \|S^{-1}MS\|_2 = \|D\|_2 = \lambda^M_{max}.$$

It follows immediately that

$$\left\| \begin{pmatrix} \frac{1}{L}\mathbf{E}[\delta_k] \\ \mathbf{E}[\|x_k - x_*\|] \\ \sigma_k \end{pmatrix} \right\|_S \le (\lambda^M_{max})^k \left\| \begin{pmatrix} \frac{1}{L}\delta_0 \\ \|x_0 - x_*\| \\ \sigma_0 \end{pmatrix} \right\|_S \tag{3.27}$$

{rconv}

where λ^M_{max} satisfies (3.22). □

It is worth noting that the above analysis provides a simple proof of the R-linear convergence of the SAG algorithm, since SAG is equivalent to Algorithm 1 applied to a finite population,with $t_k = m$, $u_k = 0$ and s_k is a constant k.

{thm:sag}

Corollary 4. *Suppose $\{x_k\}$ is produced by the SAG method applied to a population of size m, i.e. Algorithm 1 where $s_k = 1$ and $u_k = 0$ for all k, with y_k is computed with* (2.8)*. Suppose Assumptions A.2 and A.3 hold. Then, $\{\mathbf{E}\|x_k - x_*\|\}$ converges to zero R-linearly.*

Proof. Since for this problem $F(x) = \frac{1}{m}\sum_{i=1}^m f^i(x)$, we have $\sigma_k = 0$ for all k. Using similar arguments as in the proof of Lemma 1, one can show that

$$\begin{pmatrix} \frac{1}{L}\mathbf{E}[\mathbf{E}_k[\delta_k]] \\ \mathbf{E}[\|x_k - x_*\|] \end{pmatrix} \le M \begin{pmatrix} \frac{1}{L}\mathbf{E}[\delta_{k-1}] \\ \mathbf{E}[\|x_{k-1} - x_*\|] \end{pmatrix} \tag{3.28}$$

{sagrecur}

where

$$M = \begin{pmatrix} (1-\eta)(1+\alpha L) & (1-\eta)\alpha L \\ \alpha L & 1 - \alpha\mu \end{pmatrix}. \tag{3.29}$$

{M2}

Then, following similar arguments as for Lemma 2, if we use the steplength

$$\alpha = \min\left\{\beta\left(\frac{2\eta}{1-\eta}\right)\frac{\mu}{L}\frac{1}{\mu+L}, \frac{1}{L}\right\}, \quad \text{for some } \beta \in (0,1), \tag{3.30}$$

{alphasag}

then the spectral radius $\lambda_{\max}^{M}$ of the matrix M satisfies

$$\lambda_{\max}^{M} \leq 1 - \frac{\alpha\left(2\eta\mu + \alpha(\eta-1)L^2 + \alpha(\eta-1)\mu L\right)}{\alpha((\eta+1)\mu - (1-\eta)L) + 2\eta} < 1 \tag{3.31}$$

{sagspec}

It follows from (3.28) that $\{\mathbf{E}\|x_k - x_*\|\}$ converges to zero R-linearly.

□

In the final remark of this section, we specify the convergence rate of our algorithm. The result is similar to SAG for a large enough growth rate, which also depends on the condition number like $\Theta(\kappa^{-1})$, see [14].[Figen: In fact, our line of analysis provides the same rate as the SAG paper for the quadratics, perhaps can be added to the appendix.]

{lemma:order

Remark 5. *If the parameter η satisfies $\left(\frac{1-\eta}{1+\eta}\right) \leq \left((1+\kappa)\frac{\theta}{\beta}+1\right)^{-1}$, then we have*

$$\lambda_{\max}^{A} < -\frac{(1-\beta)\theta}{(1+\beta)(\kappa+\theta)} = -\Theta(\kappa^{-1}). \tag{3.32}$$

{eq:rate1}

Otherwise,

$$\lambda_{\max}^{A} < -\frac{\beta(1-\beta)(1-\left(\frac{1-\eta}{1+\eta}\right))^2}{\left(\frac{1-\eta}{1+\eta}\right)\kappa(\kappa+1)+\beta} = -\Theta(\kappa^{-2}). \tag{3.33}$$

{eq:rate2}

Proof. In (3.25), we place the corresponding value of α from (3.21):

Case I. $\alpha = \dfrac{\theta}{L}$.

$$-\frac{\alpha\mu}{1+\alpha\mu}\left(1-\left(\frac{1-\eta}{1+\eta}\right)[1+\alpha L(1+\kappa)]\right) = -\frac{\theta}{\kappa+\theta}\left(1-\left(\frac{1-\eta}{1+\eta}\right)[1+\theta(1+\kappa)]\right)$$

In light of (3.24),

$$1-\left(\frac{1-\eta}{1+\eta}\right)[1+\theta(1+\kappa)] \geq 1 - \frac{\beta(1+\theta(1+\kappa))}{\beta+\theta(1+\kappa)} = \frac{(1-\beta)\theta(1+\kappa)}{\beta+\theta(1+\kappa)} > (1-\beta)\frac{1}{\beta+1}.$$

Case II. $\alpha = \beta\left(\dfrac{2\eta}{1-\eta}\right)\dfrac{\mu}{L}\dfrac{1}{\mu+L}$.

$$\begin{aligned}-\frac{\alpha\mu}{1+\alpha\mu}\left(1-\left(\frac{1-\eta}{1+\eta}\right)[1+\alpha L(1+\kappa)]\right) &= -\frac{\beta(1-\left(\frac{1-\eta}{1+\eta}\right))(1-\left(\frac{1-\eta}{1+\eta}\right))(1-\beta)}{\beta(1-\left(\frac{1-\eta}{1+\eta}\right))+\left(\frac{1-\eta}{1+\eta}\right)\kappa(\kappa+1)} \\ &< -\frac{\beta(1-\beta)(1-\left(\frac{1-\eta}{1+\eta}\right))^2}{\beta+\left(\frac{1-\eta}{1+\eta}\right)\kappa(\kappa+1)}.\end{aligned}$$

□

4 Implementation of the EGR method

{implementat

In this section we describe interesting choices for the sequences $\{s_k\}$ and $\{u_k\}$, and the characteristics of the resulting methods. We also discuss implementation details for Algorithm 1, and make a few comments about techniques for reducing storage.

There are some implicit restrictions on the input choices of Algorithm 1. Clearly, we must have that $s_0 = 0$, $u_0 > 0$, and $s_t \leq t_k$ for all k. By construction, $t_k = \sum_{i=0}^{k-1} u_i$ for $k > 0$. For finite datasets of size m, u_k must become 0 once the whole training set has been exhausted. Table 1, lists five classes of methods that can be generated by the EGR framework, some of which are well known. A short discussion of these algorithms is given below.

Algorithm	u_k	t_k	$\frac{u_k}{t_k}$	s_k for $k > 0$	$\frac{s_k}{t_k}$
Only-add	*	*	*	0	0
Only-update	m if $k = 0$ 0 if $k > 0$	m	0	*	*
EGR-lin	r	rk	$\frac{1}{k}$	r	$\frac{1}{k}$
EGR-quad	$r(k+1)$	$\frac{rk(k+1)}{2}$	$\frac{2}{k}$	rk	$\frac{2}{k+1}$
EGR-exp	1 if $k = 0$ $r\,(1+r)^{k-1}$ if $k > 0$	$(1+r)^{k-1}$	r	$r\,(1+r)^{k-1}$	r

Table 1: Methods obtained from Algorithm 1 by specifying the sequences $\{s_k\}$ and $\{u_k\}$. A * means that any (integer) value can be chosen, and r is a given constant. Since u_k, s_k and t_k must be positive integers, the `ceiling` function $\lceil \cdot \rceil$ should be applied as needed, but we ommitted it to avoid clutter.

{tab1}

`Only-add algorithms` do not recompute old gradients. They include the stochastic gradient method ($u_k = 1, s_k = 0$) and dynamic sampling methods, where $s_k = 0$ and u_k typically grows as a function of k. Both methods are successful in practice and the behavior of the sequence $\{F(x_k)\}$ to F^* is well understood.

`Only-update algorithms` do not sample new gradients. The classic gradient descent algorithm, SAG, and SAGA fall in this category. These algorithms work on a static batch of size m and their convergence results apply to the sequence $\{\psi(x_k)\}$ defined in (2.1).

`egr-lin` is the version of the EGR method that employs a constant number r of replacements and a constant number of new samples at each iteration, which give rise to a linear growth in t_k.

`egr-quad` lets the number of replacements and new samples to grow linearly at each iteration. This results in a quadratic growth in t_k.

`egr-exp` forces the sequence $\{t_k\}$ to grow by a fixed factor at every iteration. This strategy starts with an initial batch of size 1 for u_0, and at each iteration resamples a fixed percentage of the number of seen gradients and samples an equivalent number of unseen gradients, thus leading to an exponential growth rate of t_k.

As stated, Algorithm 1 computes a sum over all $\{1, \ldots, t_k\}$ at every iteration; see (2.8) and (2.11). This costly computation can be avoided by storing the current sum, and updating it accordingly at every iteration. An efficient implementation of the EGR method is discussed in [?] and the corresponding software, written in Julia, is available at `https://github.com/stefanks/EGR.jl`.

[JN: I am leaving the following algorithm here for now. If the paper gets too long, we should simply refer to Stefan's thesis (as I wrote in the above text)]

Algorithm 2 EGR Algorithm (Implementation Version) {alg1-IMP}

```
Input: $\{s_k\}$, $\{u_k\}$, $\alpha$, $x$
$T = 0,\ k = 0,\ A = 0$
loop
    $U = \{T+1, \dots, T+u_k\}$
    For all $j \in U$ draw $\xi^j$ from the distribution $P$
    $S =$ sample of $s_k$ indices from $\{1, \dots, T\}$ uniformly
    $B = \sum_{j \in S} g^j$
    $g^j = \nabla f^j(x)$ for all $j \in S \cup U$
    $y = \frac{\frac{s_k}{T} A - B + \sum_{j \in S \cup U} g^j}{s_k + u_k}$ EGR-SAGA  or  $y = \frac{A - B + \sum_{j \in S \cup U} g^j}{T + u_k}$ EGR-SAG    (4.34)
    $x = x - \alpha y$
    $T = T + u_k$
    $A = A - B + \sum_{j \in S \cup U} g^j$
    $k = k + 1$
end loop
```

4.1 Memory Reduction Techniques

{memred}

We now discuss techniques that can be employed to reduce storage in the aggregated gradient methods, and by extension in the EGR method.

SAG and SAGA require storing m gradients to compute each iteration. In many machine learning applications, the sample gradients can be stored compactly [14]. Specifically, in machine learning applications, where the sample functions f^i are loss functions, they often have the property that

$$f^i(x) = g(W_i x), \tag{4.35}$$

{goodForComp

for some function g and $c \times n$ matrices W_i. In this case, the gradient of f with respect to x is computed as

$$\nabla f^i(x) = W_i^T g'(W_i x), \tag{4.36}$$

where $g'(W_i x)$ is a vector of length c. Therefore, storing the vector $g'(W_i x)$ is sufficient to reconstruct the original gradient at x. For binary classification problems with certain loss functions, such as the cross entropy loss, the matrix W_i is an $1 \times n$ vector, reducing the storage requirement to a scalar. Note that the addition of an $L2$ regularizer destroys the property (4.35), thereby making compact storage impossible.

We propose an alternative way of reducing this memory requirement. The strategy, which we call *chunking*, can be extended to the EGR algorithm. We expect performance to deteriorate compared to a full memory implementation (due to more expensive iterate computation), with the advantage of having a slimmer memory requirement.

Assume that s_k and u_k are divisible by some C for every k. In this case, we can introduce the concept of chunking, or mini-batch replacement and storing. This requires grouping the sample functions into chunks of size C, indexed by j. Instead of storing the values of each $\nabla f^i(\phi_i^k)$, we store appropriate sample gradient averages $\frac{1}{C}\sum_{i \in Chunk_j} \nabla f^i(x)$. Since an average of sample gradients is not of the form (4.35), the compact storage property cannot be used. On the other hand, for large enough chunks this might not be a drawback at all, since the sheer reduction in the number of vectors to be stored may make the storage requirement less demanding even compared to using compact forms. Clearly, chunking is not appropriate when storing averages of gradients is prohibitively expensive.

Note that chunking and *batching* are different algorithms. Specifically, a batched version of EGR would simply be a version with s_k or u_k being greater than 1, without reducing memory requirements.

A description of the chunking idea for the original SAG and SAGA algorithms, as well as to the proposed EGR algorithm is given in [**?**].

5 Numerical Tests

{numerical}

The purpose of this section is to qualitatively measure the performance of EGR in the early iterations of the algorithm, when the number of evaluated sample gradients is smaller than the number of available training points m, i.e. the single-epoch mode. This experimental mode allows us to make conclusions about the performance of our algorithm when used on a problem with a very large training set, and the "oracle" or "streaming" case, where new samples from P are available on demand. Both cases are common in practice. Therefore, a direct comparison of EGR with aggregated gradient methods such as SAG and SAGA is not appropriate, since any iteration of these methods requires pre-computed sample gradients ∇f^i for each function f^i where $i \in \{1, \ldots, m\}$. Instead, we test different growth sequences of EGR, including the Stochastic Gradient (SG) and the Dynamic Sample Size (DSS) methods, both special cases of EGR.

The EGR algorithm closely resembles the so-called *initialization heuristic* of SAG. This heuristic is mentioned in [4] alongside its full counterpart, as a way to gradually grow the stored number of datapoints in SAG. No theoretical support is given, and the paper claims that simply running SG instead gives nearly identical performance. A possible explanation is that the function value is not evaluated at the end of the initialization phase, but instead, the performance of the later iterations is compared. We do not claim to dismiss this statement: rather, we draw new conclusions on the efficacy of using this heuristic as a stand-alone algorithm.

The `SAG-init` method, given as Algorithm 3, works as follows: at each iteration a randomly sampled datapoint is chosen from the full dataset, the corresponding sample gradient is evaluated, and this gradient replaces the old one only if it has been sampled previously. Otherwise the new gradient is stored, and the memory grows, see [14] for details. The only difference between this method and EGR is that in our framework we pre-determine the sampling from the seen and unseen sets by the sequences u_k and s_k. In `SAG-init`, the choice of sampling from S_k or U_k is random. `SAGA-init` is similar, except that it uses the SAGA direction computation.

Algorithm 3 SAG-init and SAGA-init

{alg:sagInit

```
Input: Steplength $\alpha$, and initial iterate $x_0$
Set $T_{-1} = \{\}$, $t_{-1} = 0$
loop $k = 0, 1, 2, , \ldots$
    Draw $j$ from $\{1, \ldots, m\}$
    Evaluate $\nabla f^j(x_k) = \nabla f(x_k, \xi_j)$
    if $j \notin T_{k-1}$ then
        $T_k = T_{k-1} \cup j$
        $t_k = t_{k-1} + 1$
    else
        $T_k = T_{k-1}$
        $t_k = t_{k-1}$
    end if
    Compute $y_k = \frac{1}{t_k}\left[\nabla f^j(x_k) - \nabla f^j(\phi^j_{k-1}) + \sum_{i \in T_k} \nabla f^i(\phi^i_{k-1})\right]$    (SAG version)
    or      $y_k = \nabla f^j(x_k) - \nabla f^j(\phi^j_{k-1}) + \frac{1}{t_k}\sum_{i \in T_k} \nabla f^i(\phi^i_{k-1})$    (SAGA version)

    $$\text{Set } \phi^i_k = \begin{cases} x_k & \text{for } i = j \\ \phi^i_{k-1} & \text{otherwise} \end{cases} \qquad (5.37)$$

    $x_{k+1} = x_k - \alpha y_k$
end loop
```

The Streaming SVRG method [?] would fit ideally in our experimental setup, since the motivation is identical to EGR. Unfortunately, we were not able to select parameters that would make this algorithm competitive. Scaling algorithms such as SQN [?] and ADA-GRAD [?], while having superior performance, are outside the scope of this project.

The testing setup is as follows: all problems are publicly available binary classification datasets. Each dataset is split into two parts: a *training set* and a *testing set.* The training set constitues the information available to the algorithms through the iterations, and is used as a proxy for sampling from the distribution P. The testing set is used to assess the effectiveness of the algorithms compared, which we do using the *test function value*, which is defined as (2.1) using the testing set. Training points constitute 75% of the total number of datapoints. We use a cross-entropy loss function with L2 regularization. The L2 regularization parameter is always $\frac{1}{\text{num training points}}$.

The basis for comparison of the different methods is the number of gradient evaluations: this is a fair estimate of the total work expended for the methods tested.

5.1 Test Problems

The datasets are all dense, with approximately evenly split -1 and +1 labels. The dataset dimensions are summarized in Table 2.

- **alphaGood** is a scaled version of the alpha dataset from `http://leon.bottou.org/projects/sgd`. The features are scaled to have zero mean and unit variance
- **MNIST** is a popular digit recognition dataset.

Dataset	Num Features	Num Datapoints
alphaGood	500	500,000
MNIST	768	60,000
myrand	50	700,000
SUSY	18	5,000,000

Table 2: Dataset statistics

{table:sizes

- **myrand** is an artificial dataset generated from two different multinomial Gaussian distributions.

- **SUSY** is a classification dataset from `https://archive.ics.uci.edu/ml/datasets/SUSY`, with the positive label signifies particle detection.

5.2 Results

We conducted three sets of experiments, each with a different objective.

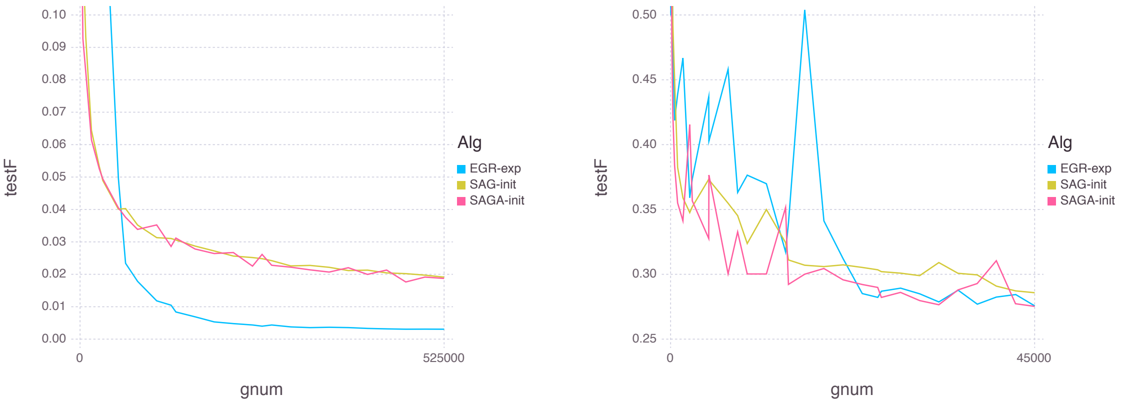
- **Performance Comparison** A comparison of a reasonably chosen EGR implementation with some competing algorithms such as SAG-init, SAGA-init, and SG.

- **EGR Benchmarking** EGR growth rates comparison. We test the empirical performance of the three suggested growth rates. Suggestions as to the best choice of parameters for any given problem are made. The two EGR versions, EGR-SAG and EGR-SAGA are also compared. Stepsize tuning to different goals is discussed in detail.

- **DSS Tests** We test the relative performance of our methods with the DSS method, when identical growth rates are used. A discussion of the benefits of using/updating history concludes the section.

5.2.1 Performance Comparison

We compare the EGR method with two other similar algorithms: The SAGinit and SAGAinit methods are not commonly thought of as optimization methods, but are presented as initialization heuristics in the corresponding papers. Nevertheless, the methods are considered as a way to run these aggregated gradient methods while building up memory as you go along. The EGR framework includes algorithms which outperform these heuristics: they are able to consistently reach a better F value, see Figure 1.

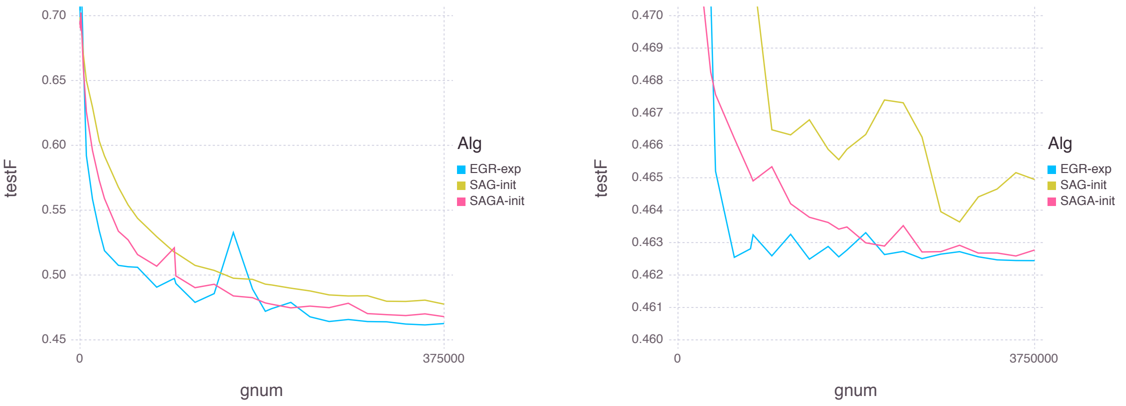


(a) myrand

(b) MNIST

Figure 1: EGR vs other methods

{fig:myrando



(a) alphaGood

(b) SUSY

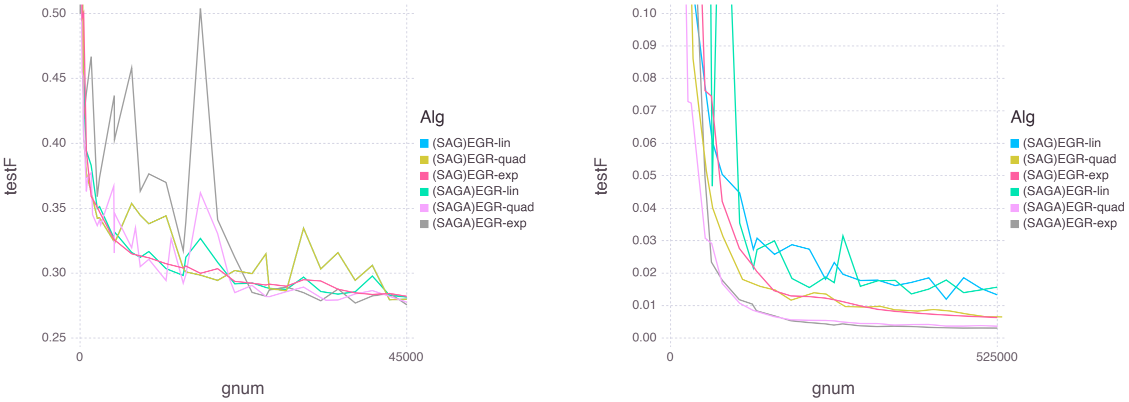
Figure 2: EGR vs other methods

{fig:alphaGo

5.2.2 EGR Benchmarking

The three different growth rates (linear, quadratic, exponential) combined with two possible computation formulas for y_k give rise to six methods, where each growth has an unspecified parameter. We tune it experimentally for each problem, and make observations of the effect of varying r in each case. Finally, we make suggestions on good practical choices or r.

In all plots in this section, the vertical axis is the test function value, which serves as an approximation to F. We demonstrate the continuous progression of the algorithms toward the minimizer of F. The horizontal axis is the total number of sample gradients evaluated so far. This is an approximate measure of the total work the algorithm has performed, and is more informative than the more commonly used plots where the horizontal axis is the iteration counter. All experiements were stopped when the number of sample gradients evaluated reached the size of the complete training set. Note that for our choices of the growth rates, when $s_k = u_k$, this means that roughly half of the avaiable training points was seen by the algorithm. The last remaining parameter, α, was chosen to give the best testing function value at the end of the training epoch. A complete collection of these experiments for each of the six methods, on all four test problems can be seen in Figures 9, 11, 10, and 12. These extensive experiments are summarized in Figures 3, 6, **??**, and **??** respectively. These summary plots show the best performing growth rates compared against each other.

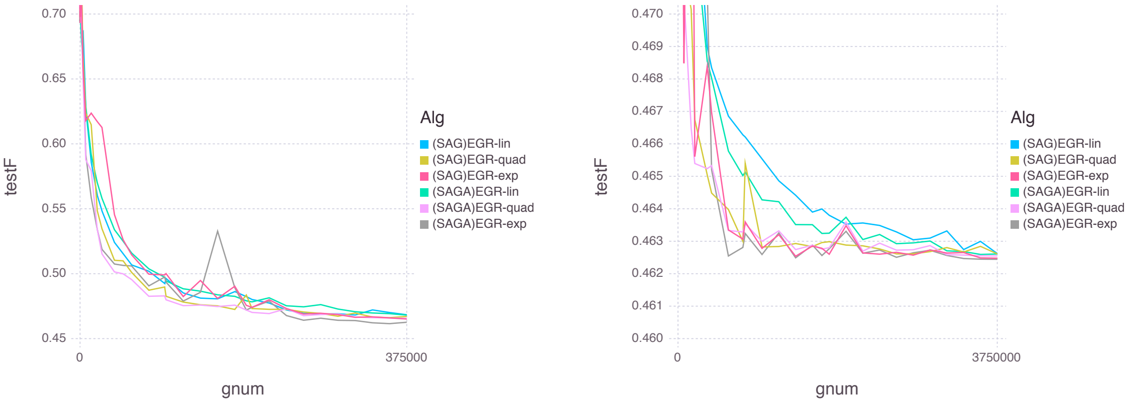


(a) MNIST

(b) myrand

Figure 3: summary: all EGR methods

{fig:MNISTsu



(a) alphaGood

(b) SUSY

Figure 4: summary: all EGR methods

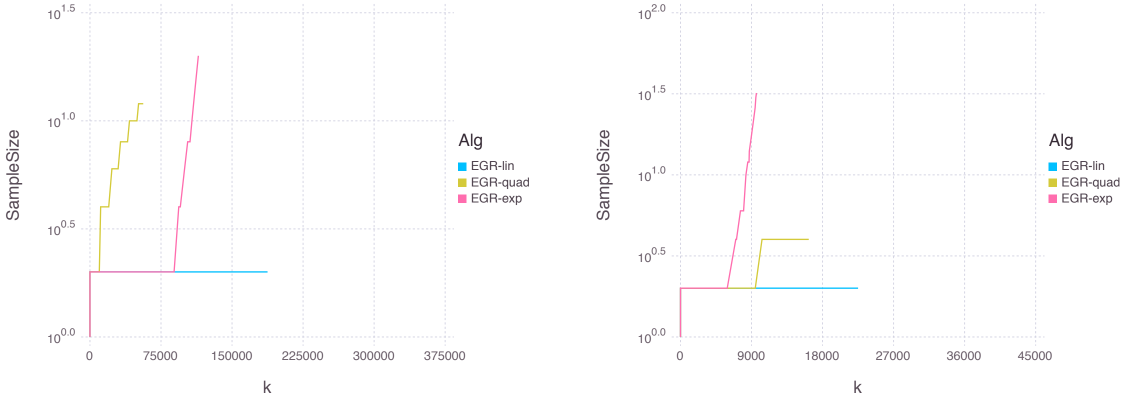
{fig:alphaGo

On problem MNIST, in Figure 9, we see that for EGR-lin(SAG) smaller batch sizes are preferable. When $r = 1$, the method performs extremely well, but slightly increasing the batch to $r = 10$ allows for significantly larger stepsizes to be taken. This closely follows the common knowledge about the need for mini-batching in SG methods. As r increases, larger steplengths are needed for good performance: this is logical since the methods use more new information at each iteration, thus having less variance in the step directions. Large batch sizes do not work well, since the method is only able to take a few iterations in the allotted sample gradient evaluation limit. The EGR-lin(SAGA) version shows similar patterns, but a higher variance in the algorithm progression is seen. This is due to the fact that the SAG version of the algorithm places a much larger emphasis on the relatively stable term involving the summation of historic sample gradient information (which also implies that smaller stepsizes than in EGR-lin are needed). In the quadratic EGR-SAG growth rate case, the smallest r tested gives the same method as the linear $r = 1$ method because of rounding. The second growth rate (corresponding to $r = 0.0001$) behaves identical to the smallest one initially, but approximately after the first half of the graph, the two plots diverge. This is because with the second growth rate the batch sizes start increasing from $s_k = u_k = 1$. In the quadratic EGR-SAGA case, this sudden increase of the sample sizes enables the superior performance of the method. For the exponentially growing case, the same idea as before applies: A slightly increasing batch size in the latter iterations seems to benefit the stability and sometimes the performance of the methods.

In Figure 3, we combine these methods in two plots. The one on the left always shows the true progression of the algorithm to the minimizer, i.e. the current estimate of the function value of F. The plot on the right shows the same data plotted to display the current best solution. The EGR-SAGA methods with growing batch sizes at the end seem to work best in practice. We attribute this to the unbiased nature of the methods. We see that the best performing methods are the ones which increase the sample sizes at the end, instead of leaving them at a constant level.

With the myrand dataset, the picture is slightly different. In Figure 10, we can see that the required growth rates are higher than for MNIST. In Figure **??**, we again see the clear advantage of using growing batch sizes, and of SAGA step computation formula over the SAG one.

On the alphaGood dataset, and the SUSY dataset, the conclusions are similar. In Figure **??** we provide a plot of current sample size, $s_k + u_k$, at each iteration of the algorithm, on the MNIST dataset. The best performing growth rate choice seems to be the one where approximately for this first half of the algorithm progression the sample sizes are as small as possible, and only then start growing.

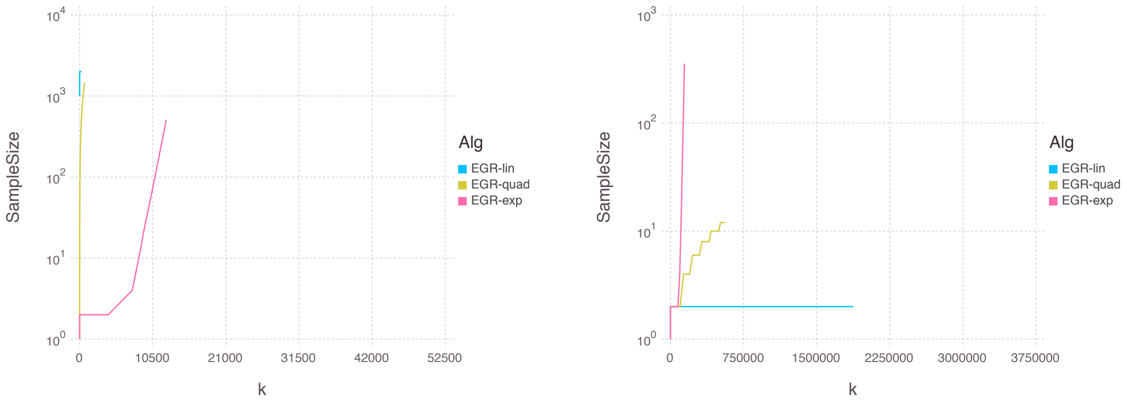


(a) alphaGood

(b) MNIST

Figure 5: growth rates

{fig:alphaGo



(a) myrand

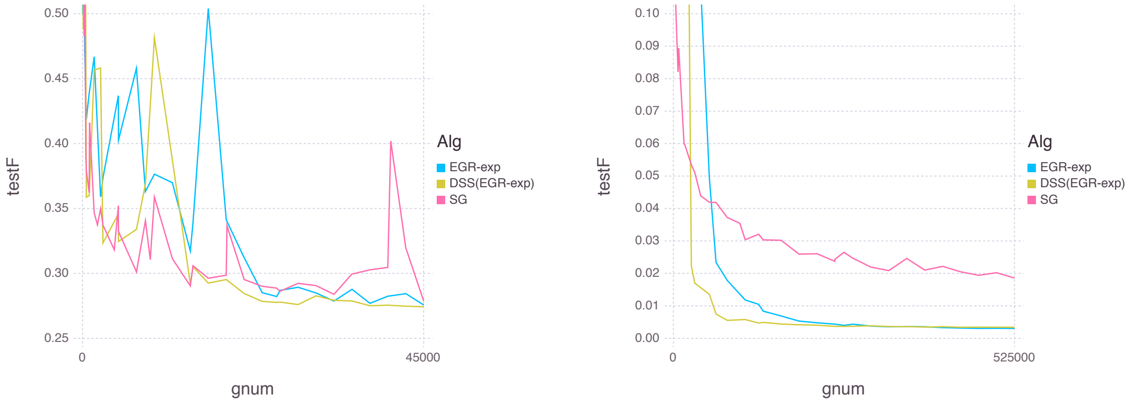
(b) SUSY

Figure 6: growth rates

{fig:alphaGo

5.2.3 Only Add Comparison

The only-add option in EGR includes many existing methods, such as SG and DSS. We empirically compare the performance of our new methods with SG and DSS. The Stochastic Gradient method enjoys immense popularity in both the optimization and the machine learning communities, and when carefully tuned can outperform any known modern method. The tuning often involves intricate step size strategies. We do not attempt to compare with the best SG implementation, rather we demonstrate the benefits of using the EGR scheme, compared to SG, with both methods using a constant step size. The DSS method tested uses a sample size identical to the one suggested by EGR, to make the per-iteration comparison fair. The plots for each problem can be found in Figures 7, **??**, **??**, 8.

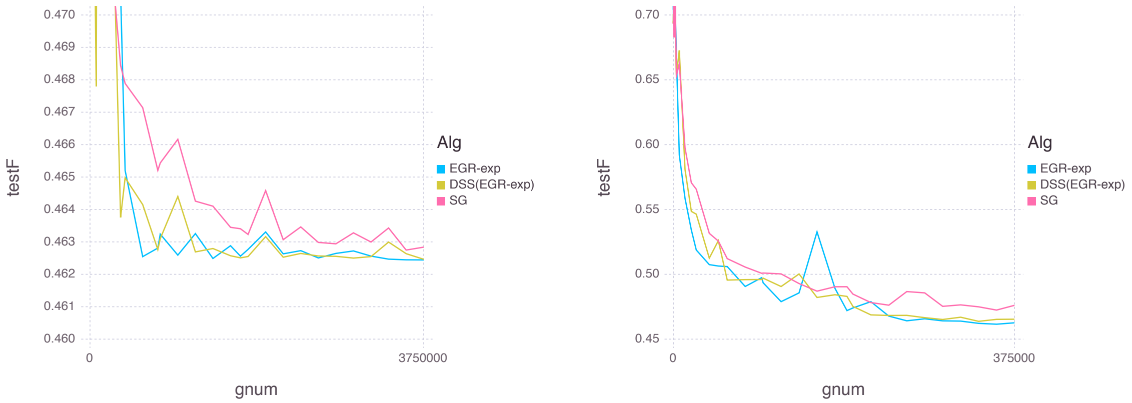


(a) MNIST

(b) myrand

Figure 7: EGR vs only-add methods

{fig:MNISToa



(a) SUSY

(b) alphaGood

Figure 8: EGR vs only-add methods

{fig:SUSYoa}

These plots show that the benefit of having multiple sample gradients evaluated at some iterations is advantageous to having a single one being evaluated at each step. In most of our experiments, DSS performed slightly better, because it was able to see all data while EGR only half. This advantage is not uniform, and we cannot make a justifiable recommendation of using one over the other.

5.2.4 Memory Reduction Techniques in Practice

The following experiments show the algorithm progress after 40 passes over the data. Only two types of chunking are tested: SAG and SAGA chunking. The labels are in the format `algorithm numChunks StepsizePower`. The stepsizes were tuned to give the best final function value.

The follwing experiements show comparisons of the aggregated gradient methods SAG and SAGA with respect to Batching vs Chunking. We conducted experiments comparing Chunking vs Batching on three datasets, and two aggregated gradient methods: SAG and SAGA. This experiment was suggested to verify the previous chunking results (on pages 41-43), because these previous experiments suggested that the chunking idea is not working well in practice.

The new experiments are ran with 40 equivalent passes over the data, and stepsizes were chosen to give the best possible function value over the progress of all iterations. Each method was tested with the following two configurations: a) Batching - batches of size 2, chunks of size 1 (blue and red on plots) b) Chunking - chunks of size 2, batches of size 1 (yellow and green on plots)

The chunking methods seem to perform much worse than the equivalent batching methods.

6 Final Remarks

{finalr}

References

[1] Reza Babanezhad, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečný, and Scott Sallinen. Stop wasting my gradients: Practical SVRG. *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[2] Dimitri P Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, pages 1–38, 2010.

[3] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample size selection in optimization methods for machine learning. *Mathematical Programming*, 134(1):127–155, 2012.

[4] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc., 2014.

[5] Aaron J Defazio, Tibério S Caetano, and Justin Domke. Finito: A faster, permutable incremental gradient method for big data problems. *arXiv preprint arXiv:1407.2710*, 2014.

[6] M.P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *Arxiv preprint arXiv:1104.2373*, 2011.

[7] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. *arXiv preprint arXiv:1412.6606*, 2014.

[8] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[9] Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855, 2015.

[10] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Math. Program.*, 120(1):221–259, 2009.

[11] R. Pasupathy, P. W. Glynn, S. Ghosh, and F. Hahemi. How much to sample in simulation-based stochastic recursions? 2014. Under Review.

[12] B.T. Polyak and A.B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838, 1992.

[13] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

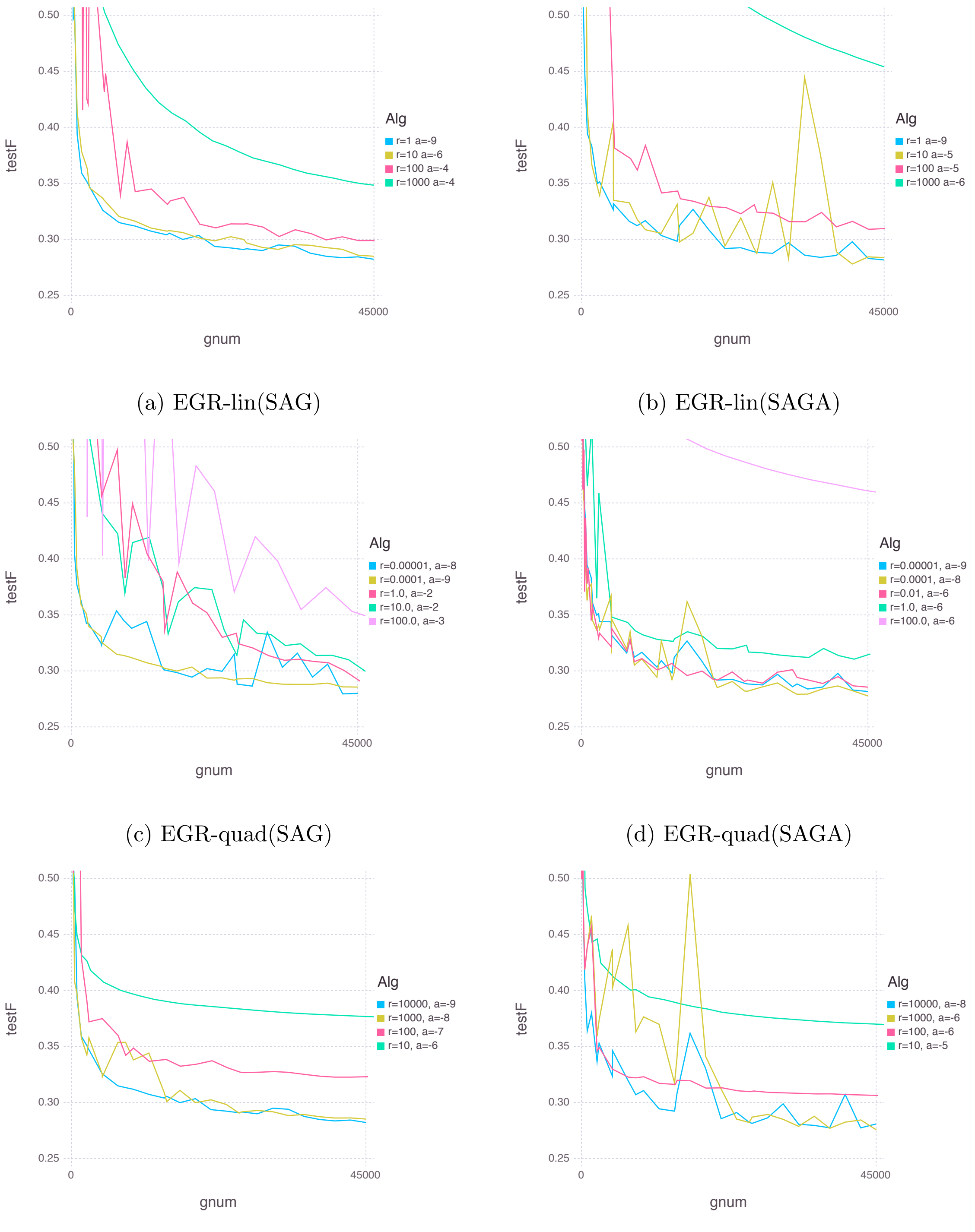
[14] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[15] David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[16] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

A EGR Growth Rates Detailed

{appendix:EG



(a) EGR-lin(SAG)

(b) EGR-lin(SAGA)

(c) EGR-quad(SAG)

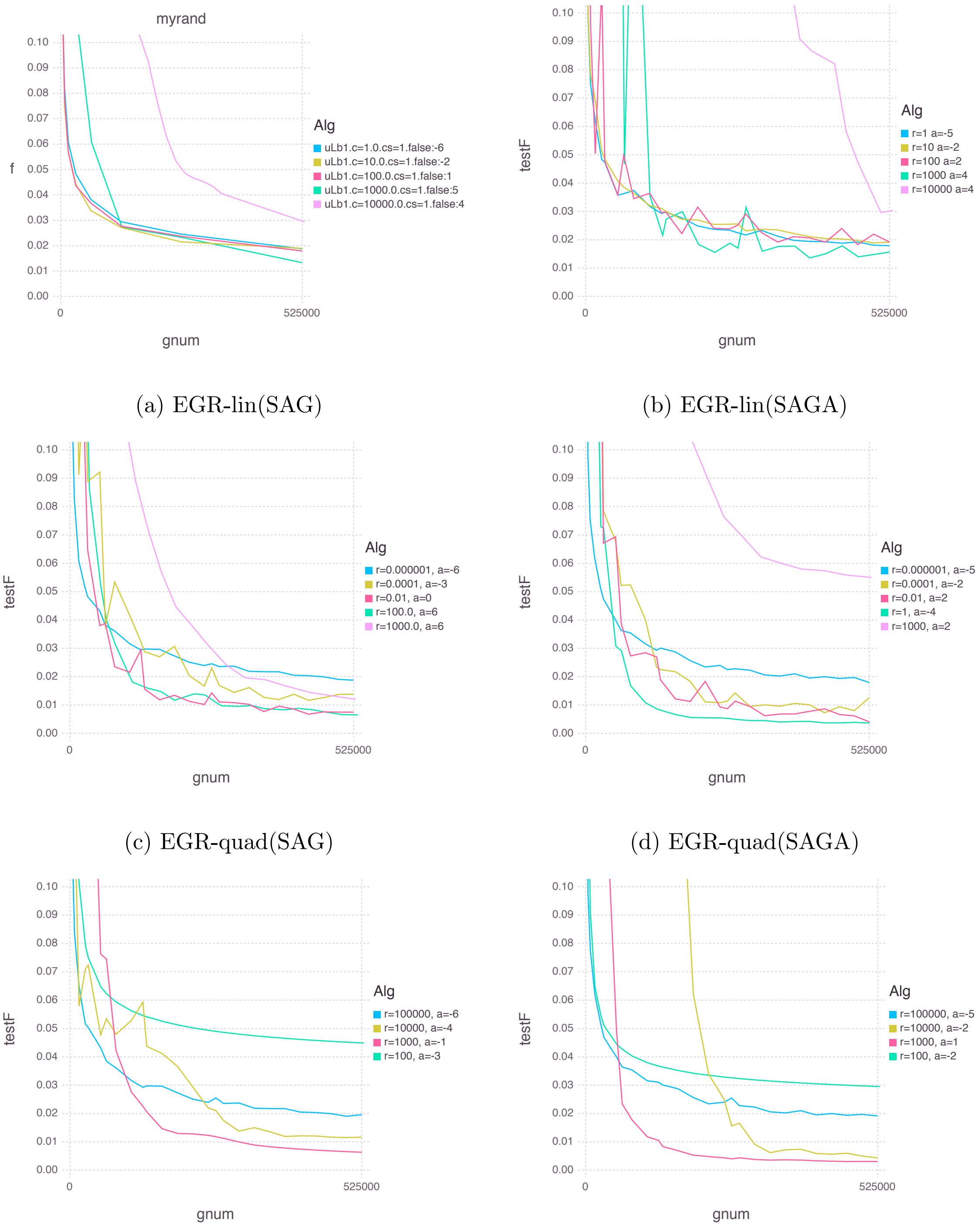
(d) EGR-quad(SAGA)

(e) EGR-exp(SAG)

(f) EGR-exp(SAGA)

Figure 9: MNIST detailed: Comparing various growth rates. In the legend, r is the parameter from Table 1 and $a = log_2(\alpha)$ where α is the constant stepsize parameter.

{fig:MNIST}



(a) EGR-lin(SAG)

(b) EGR-lin(SAGA)

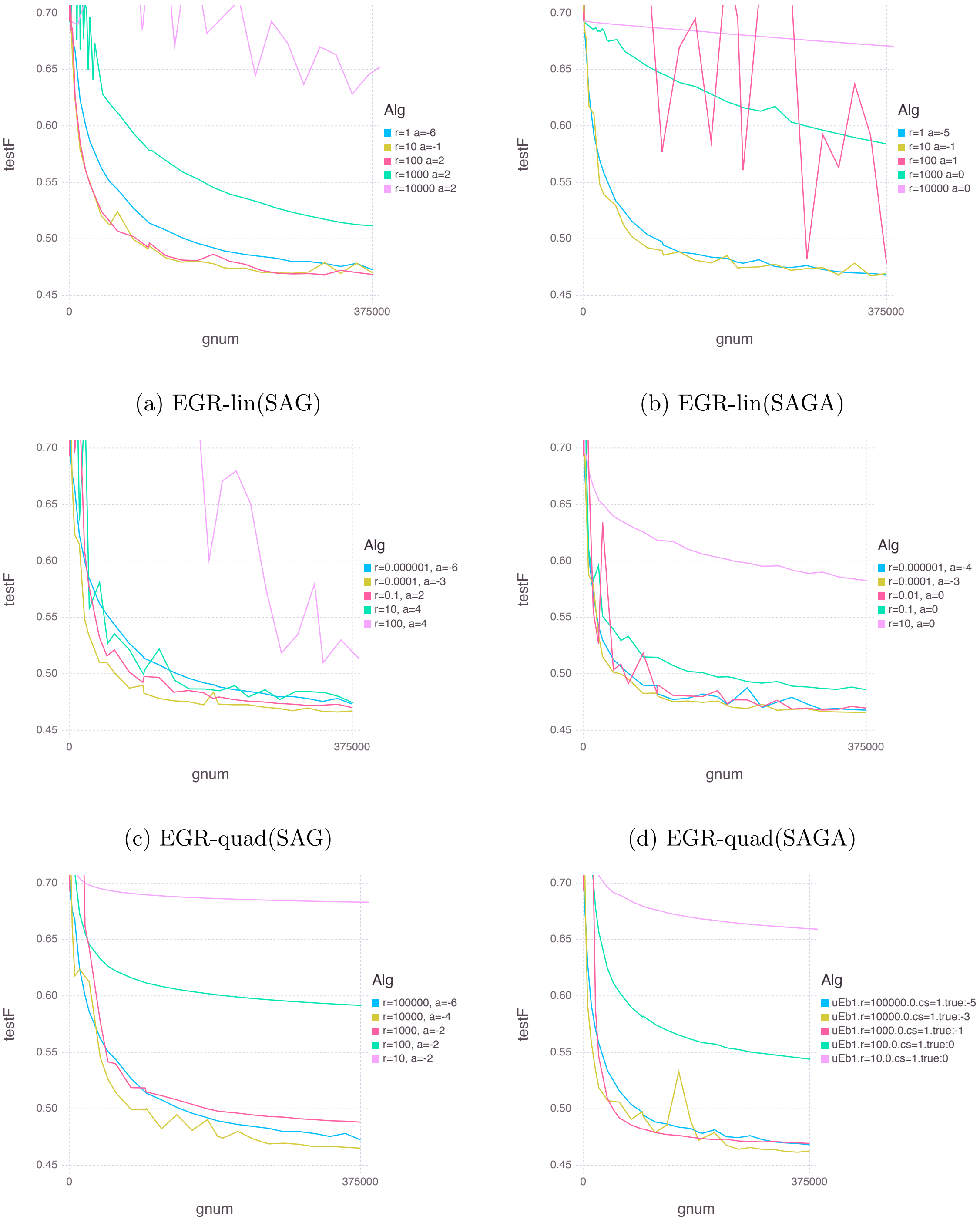
(c) EGR-quad(SAG)

(d) EGR-quad(SAGA)

(e) EGR-exp(SAG)

(f) EGR-exp(SAGA)

Figure 10: myrand detailed: Comparing various growth rates. In the legend, r is the parameter from Table 1 and $a = log_2(\alpha)$ where α is the constant stepsize parameter. {fig:myrand}



(a) EGR-lin(SAG)

(b) EGR-lin(SAGA)

(c) EGR-quad(SAG)

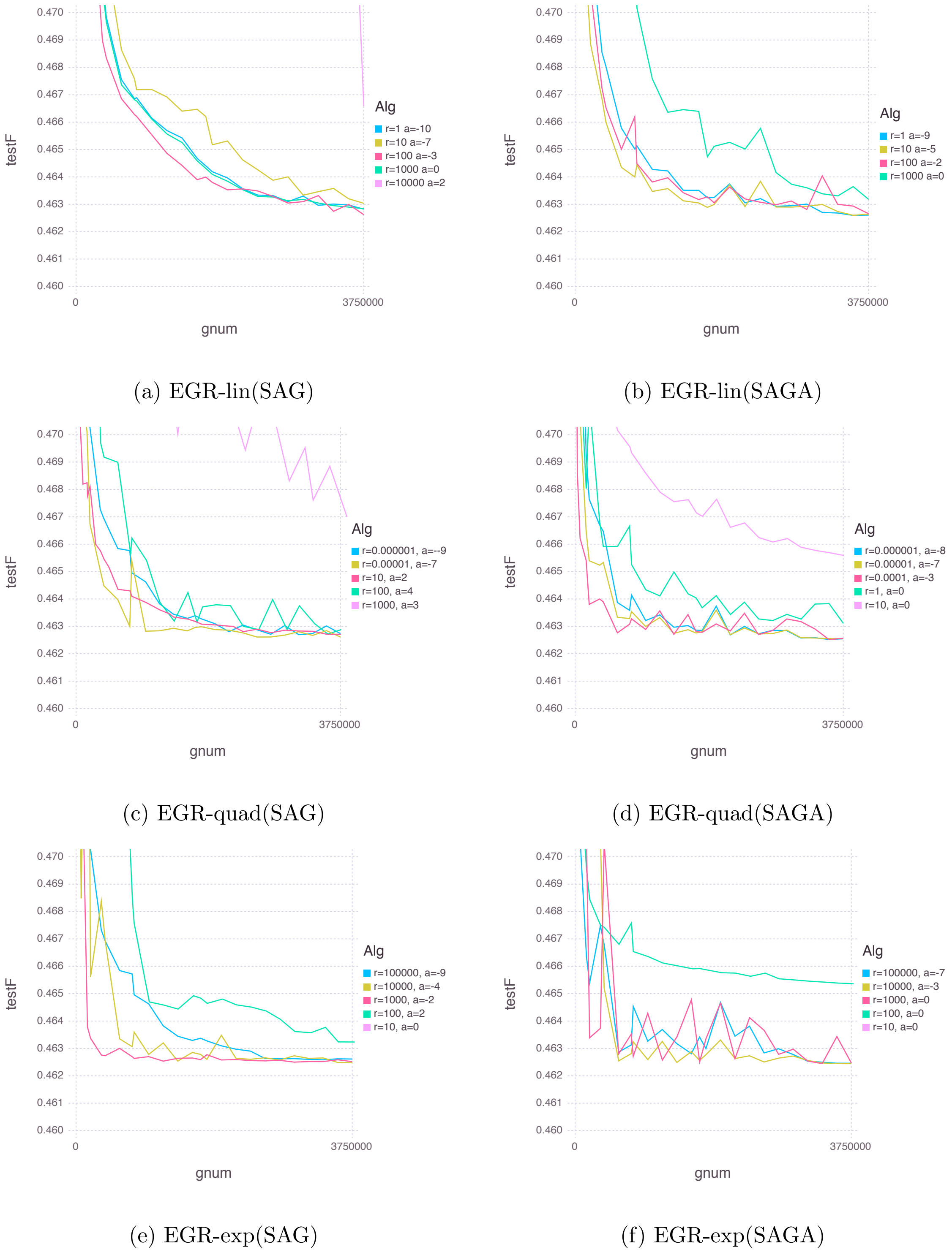
(d) EGR-quad(SAGA)

(e) EGR-exp(SAG)

(f) EGR-exp(SAGA)

Figure 11: alphaGood detailed: Comparing various growth rates. In the legend, r is the parameter from Table 1 and $a = log_2(\alpha)$ where α is the constant stepsize parameter.

{fig:alphaGo



(a) EGR-lin(SAG)

(b) EGR-lin(SAGA)

(c) EGR-quad(SAG)

(d) EGR-quad(SAGA)

(e) EGR-exp(SAG)

(f) EGR-exp(SAGA)

Figure 12: SUSY detailed: Comparing various growth rates. In the legend, r is the parameter from Table 1 and $a = log_2(\alpha)$ where α is the constant stepsize parameter. {fig:SUSY}