Computational Optimization and Applications
https://doi.org/10.1007/s10589-026-00776-2

Effectively leveraging momentum terms in stochastic line search frameworks for fast optimization of finite-sum problems

Matteo Lapucci[1] · **Davide Pucci**[1]

Received: 11 November 2024 / Accepted: 26 February 2026

Abstract
In this work, we address unconstrained finite-sum optimization problems, with particular focus on instances originating in large scale deep learning scenarios. Our main interest lies in the exploration of the relationship between recent line search approaches for stochastic optimization in the overparametrized regime and momentum directions. First, we point out that combining these two elements with computational benefits is not straightforward. To this aim, we propose a solution based on mini-batch persistency. We then introduce an algorithmic framework that exploits a mix of data persistency, conjugate-gradient type rules for the definition of the momentum parameter and stochastic line searches. The resulting algorithm provably possesses convergence properties under suitable assumptions and is empirically shown to outperform other popular methods from the literature, obtaining state-of-the-art results in both convex and nonconvex large scale training problems.

Keywords Finite-sum optimization · Stochastic line search · Momentum · Data persistency

Mathematics Subject Classification 90C30 · 90C26 · 90C06 · 65K05 · 68T07

1 Introduction

In this paper we focus on finite-sum minimization problems where the objective function has a large number of possibly nonconvex terms. Precisely, we address the problem

✉ Davide Pucci
davide.pucci@unifi.it

Matteo Lapucci
matteo.lapucci@unifi.it

[1] Global Optimization Laboratory, Department of Information Engineering, University of Florence, Via di Santa Marta, 3, 50139 Florence, Italy

Published online: 11 March 2026

$$\min_{x\in\mathbb{R}^n} f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x), \tag{1}$$

where N is large and $f_i : \mathbb{R}^n \to \mathbb{R}$ are differentiable, possibly nonconvex functions for all $i \in \{1, \ldots, N\}$. The development of high-performance algorithms for this scenario has been driven in recent years by the explosion of deep-learning applications: indeed, supervised learning tasks are instances of problem (1).

State-of-the-art methods to tackle problems of this form are primarily based on stochastic gradient descent, SGD [1] and its accelerated variants [2, 3]. The use of stochastic gradient methods allows to leverage data redundancy to approach good solutions while carrying out low-cost iterations. A thorough discussion on stochastic gradient methods and their application in data science is available in [4].

Adaptive SGD approaches [5–8] have established through the years as solid choices in the considered scenario. In particular, Adam algorithm [8] is now largely considered the preferred optimizer in deep learning settings. In an alternative stream of research, variance-reduced approaches [9–11] have been studied and shown to possess faster convergence rates than the base algorithm and in line with full-batch gradient descent. However, in practice, these methods require significant computational resources, either needing periodic full gradient computations or high memory usage; for this reason, variance reduction methods are often found to be impractical to use in real-world scenarios with deep architectures and very large datasets. Moreover, the assumptions made to obtain the nice convergence results have been shown not to be satisfied in deep learning scenarios. In fact, no improvement in the convergence rate [12] w.r.t. SGD is obtainable under more realistic assumptions.

In more recent years, the focus turned to the analysis of stochastic gradient methods under a specific assumption that is reasonable with modern deep learning models: the interpolation regime. In this scenario, the convergence rate of SGD was finally shown to match that of full-batch methods [13, 14], giving some justification to the observed behavior of these methods. Moreover, under interpolation assumptions it was possible to design well-grounded strategies to choose the step size in an adaptive manner [15, 16].

Now, one of the most promising developments concerns the use of stochastic line searches. In particular, the classical Armijo rule was adapted to be successfully used in the incremental scenario [17]. More recently, a nonmonotone version of the stochastic Armijo line search was proposed [18]. With these techniques, not only the convergence rate of standard gradient descent is shown to be obtainable, but very encouraging numerical results were also observed.

As widely recognized in the machine learning literature, the addition of momentum terms in the update rule has been known to be beneficial within SGD approaches [19–23]; typically, this is attributed to the stabilizing effect on the direction and the speed-up in low curvature regions. It is still not clear, however, how this kind of additional term, which modifies the search direction, could be suitably handled in line search based frameworks. A heuristic choice for the step size in presence of momentum terms under interpolation assumptions has been proposed by [24], but no decrease condition is coupled with this strategy. On the other hand, a backtracking

strategy has been recently suggested, ensuring the resulting direction is of descent for the considered mini-batch [25]; for this latter approach, however, we will outline massive shortcomings both on the theoretical and on the computational sides.

In this manuscript, we therefore deal with the above challenge. Firstly, we shed light on an intrinsic issue related to the momentum direction in the incremental regime; in order to overcome it, at least partially, we propose to exploit the concept of mini-batch persistency [26], which we also observe being beneficial on its own in certain settings.

We can then focus on a suitable rule to define the parameter β associated with the momentum term in the direction update. The proposed method exploits the strong connection of momentum methods and classical nonlinear conjugate gradient algorithms [27, Chap. 12]: in both cases, the search direction is a linear combination of the current gradients and the previous direction, and the border between the two families of approaches is thin. The update rule is combined with safeguarding strategies based on restarting [28] or subspace optimization [29], that allow to guarantee crucial assumptions for the soundness of the resulting algorithm [30].

For the proposed algorithmic framework, we provide a brief discussion concerning the major, substantial challenges associated with the theoretical analysis of the proposed method—namely, the biasedness of the search direction in presence of data persistency and the relationship of the direction itself with the true gradient—and present a set of workarounds that make the algorithm provably convergent under PL condition and interpolation.

We finally present the results of in-depth computational experiments, showing that it is competitive and even outperforms in certain scenarios the main state-of-the-art optimizers for both convex (linear models) and nonconvex (deep networks) learning tasks.

The reminder of this paper is organized as follows: in Sect. 2 we recall the main concepts and algorithms to tackle finite sum problems, with a particular focus on algorithms that employ stochastic line searches under interpolation assumptions. In Sect. 3, we discuss the difficulties encountered when momentum terms are employed in combination with line searches (Sect. 3.1), proposing in Sect. 3.2 a strategy to suitably leverage momentum information. In Sect. 4 we describe some conjugate gradient rules to determine a suitable value for the parameter β associated to the momentum term, that exploit data persistency. In Sect. 5, we present our algorithmic framework, which exploits momentum terms alongside stochastic line searches. In Sects. 5.1.1 and 5.1.2 we address the main challenges involved in establishing convergence results for the proposed algorithmic framework, then reporting the main theoretical result in Sect. 5.1.3. Section 6 is devoted to computational experiments aimed at assessing the effectiveness of the proposed method. Finally, Sect. 7 provides concluding remarks.

2 Preliminaries

Problem (1) is typically solved by means of stochastic gradient descent (SGD) type algorithms. Since N is very large and the evaluation of exact derivatives is therefore expensive, at each iteration a cheap approximation of the gradient $\nabla f(x)$ is considered. The approximation is usually obtained randomly sampling a subset (referred to as

mini-batch) $B_k \subset \{1, \ldots, N\}$ and thus defining

$$f_k(x) = \frac{1}{|B_k|} \sum_{i \in B_k} f_i(x), \quad g_k(x) = \nabla f_k(x) = \frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x). \tag{2}$$

The update rule is then given by $x^{k+1} = x^k + \alpha_k d_k$, where the step size α_k is often referred to as *learning rate* and d_k is the search direction. A deep presentation of stochastic gradient methods for machine learning tasks is available in [4]. Under regularity assumptions, and using a suitable decreasing sequence of steps $\{\alpha_k\}$, the iterative scheme allows to guarantee convergence to stationarity in expectation. The associated worst-case complexity bound is $\mathcal{O}\left(\frac{1}{\epsilon^4}\right)$ in the nonconvex setting [31], which becomes $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ and $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ under convexity and strong convexity assumptions, respectively [4]. These bounds are worse than those obtainable by full-batch methods, which should be thus superior. Yet, the empirical evidence collected training deep machine learning models tells another story.

Recently, some explanation to this gap between theory and numerical observations was found: modern machine learning models are usually expressive enough to fit any data point in the training set [13, 32], i.e., the so-called *interpolation condition* holds, which means that, given $x^* \in \arg\min_{x \in \mathbb{R}^n} f(x)$, then $x^* \in \arg\min_{x \in \mathbb{R}^n} f_i(x)$ for all $i \in \{1, \ldots, N\}$. When the interpolation property holds, SGD methods can be shown to match gradient descent complexity of $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ in the convex case and its linear rate under strong convexity.

Additionally, the landscape of the training loss of deep models has been studied [33]; while convexity and strong convexity seem not to hold, not even locally, another property appears to hold in large portions of the space: the *Polyak–Lojasiewicz* (PL) condition [34]. Very interestingly, the linear convergence rate was shown to hold for SGD even with this weaker property, if interpolation is taken into account [16–18].

Interpolation implies a tighter relationship between the stochastic functions $f_k(x)$ and the true loss $f(x)$; this strong bond allowed to define more sophisticated strategies to choose the step size, with nice results both from the theoretical and the computational sides. Hereafter, we summarize some of these relevant linearly convergent approaches.

Loizou et al. [16] proposed a stochastic variant of the Polyak step size [34] for guessing the most promising step size for SGD at each iteration. In particular, the *Stochastic Polyak step size* (SPS) is obtained assuming the optimal function value for the current mini-batch objective f_k to be some known value f_k^* and reachable moving along d_k; the SPS is consequently obtained, minimizing a first-order model, setting

$$\alpha_k^{\text{SPS}} = \frac{f_k(x^k) - f_k^*}{c||\nabla f_k(x^k)||^2}, \tag{3}$$

where $c > 0$ is a normalization parameter. In the same work, the bounded variant $\text{SPS}_{\max}$ is also presented, considering $\alpha_k = \min\left\{\alpha_k^{\text{SPS}}, \alpha_{\max}\right\}$ where $\alpha_{\max} > 0$ is a

suitable upper bound for the step size. The assumption on the value of f_k^* is reasonable in most cases, as it is linked to the interpolation condition and the non-negative nature of loss functions in most machine learning applications. For instance, it can be set to 0 with unregularized least squares, or logistic losses or by some closed-form rule in the corresponding regularized cases (see, e.g., [16, Sect. 2.1]).

Inspired by the arguably top performing methods in classical nonlinear optimization, line searches have also been considered to achieve fast convergence in the interpolation regime. The first stochastic line search specifically designed for this scenario was introduced in [17]: the extension of an Armijo-type condition [35] to the stochastic case is shown to lead to the desired linear convergence rate under reasonable assumptions. The line search is structured so that the progress in the current stochastic function is measured, looking for a sufficient decrease that depends on the stochastic gradient direction; formally, the proposed Armijo condition is

$$f_k(x^k - \alpha_k g_k(x^k)) \leq f_k(x^k) - \gamma \alpha_k \|g_k(x^k)\|^2. \tag{4}$$

Checking whether this condition is satisfied or not for a given step size α is rather cheap, only needing to make an additional evaluation of the stochastic function; a suitable step size α_k can thus be easily computed by a classical backtracking procedure; possibly, we would like to select a reasonable initial step size to reduce the number of backtracks, and thus function evaluations, to the bone; for this reason, heuristic strategies have been devised to determine a good guess step.

In fact, the strict decrease on the current stochastic objective might appear as an unnecessarily strong condition to impose, especially in highly nonlinear and nonconvex problems. This observation was the main motivation, in [18], to introduce a nonmonotone variant of the Armijo type condition. Letting

$$C_k = \max\{\tilde{C}_k, f_k(x^k)\}, \quad \tilde{C}_k = \frac{\xi Q_k C_{k-1} + f_k(x^k)}{Q_{k+1}}, \quad Q_{k+1} = \xi Q_k + 1,$$

so that C_k is a value greater or equal to both a linear combination of previously computed (stochastic) function values the and the current value $f_k(x^k)$, the nonmonotone decrease condition is given by

$$f_k(x^k - \alpha_k g_k(x^k)) \leq C_k - \gamma \alpha_k \|g_k(x^k)\|^2. \tag{5}$$

The employment of this condition instead of (4) in line search based mini-batch GD might contribute to reduce the function evaluations required by repeated backtracks and to more often accept aggressive step sizes, without sacrificing convergence speed. Combined with an initial guess step of the form (3), the resulting algorithm, named PoNoS, proves to be computationally very efficient at solving nonconvex learning tasks.

3 Integrating momentum terms with SLS

It is broadly accepted that the employment of momentum terms in gradient based optimization of finite-sum functions has a considerable positive impact [4, 36]. Specifically, heavy-ball like updates follow rules of the form

$$x^{k+1} = x^k - \alpha_k g_k(x^k) + \beta_k(x^k - x^{k-1}), \tag{6}$$

where β_k is a suitable positive scalar. In the stochastic setting, partly repeating the previous update mainly helps at stabilizing the optimization trajectory and at taking larger steps in low-curvature regions. Momentum is thus employed in most state-of-the-art optimizers for large-scale nonconvex finite-sum problems, e.g., it is included in Adam [8].

It should not thus be surprising that, within the most recent lines of research concerning the selection of the step size, momentum direction ended up being considered. In this section we discuss existing strategies to integrate momentum terms in modern stochastic optimization frameworks, highlighting a major shortcoming. We then propose an approach to suitably overcome the issue.

3.1 Existing strategies and computational shortcomings

A generalization of the Stochastic Polyak step size was introduced to determine the appropriate step size when considering a momentum-type direction [24]. In particular, the step size associated with moving average of gradients (MAG) directions of the form $d_k = -(1-\beta)\nabla f_k(x^k) + \beta(x^k - x^{k-1})$, being $\beta \in (0, 1)$ a fixed parameter, is defined as

$$\alpha_k = \min\left\{\frac{f_k(x^k) - f_k^*}{c||d_k||^2}, \alpha_{\max}\right\}, \tag{7}$$

resulting in a simple, yet powerful extension of (3). Of course, an analogous rule can be applied with directions of the form

$$d_k = -g_k(x^k) + \beta_k(x^k - x^{k-1}); \tag{8}$$

yet, no hint is provided for the definition of a suitable β_k.

The combination of a momentum-type direction and the use of a stochastic line search was recently discussed [25]. The resulting algorithm, (called MSL_SGDM) revolves around directions of the form (8). In case, for a given value of β, d_k is not a descent direction for the current stochastic function f_k [i.e., $d_k^T \nabla f_k(x^k) > 0$], it is modified by either setting $\beta = 0$ or by reducing β until $d_k^T \nabla f_k(x^k) < 0$. Once d_k is ensured to be of descent, the step size α_k in the update

$$x^{k+1} = x^k + \alpha_k d_k \tag{9}$$

can be computed according to satisfy a generalized Armijo condition according to

$$\alpha_k = \max_{j=0,1,\ldots} \{\alpha_0^k \delta^j \,|\, f_k(x^k + \alpha_0^k \delta^j d_k) \leq f_k(x^k) + \gamma \alpha_0^k \delta^j d_k^T \nabla f_k(x^k)\}. \tag{10}$$

By means of the same strategy, other families of directions could be considered, as long as d_k can be manipulated to satisfy the "stochastic" descent property for the mini-batch; Adam in particular is considered in [25], but Mean Average Gradient (MAG) or Nesterov-type directions could also be handled by this framework in principle.

Although apparently well planned, the strategy from [25] hides a significant drawback, as we will argue hereafter. Surely, it is a given fact that, in the incremental setup, using a momentum-based algorithm somehow allows to mitigate the presence of noise in gradient evaluations caused by the usage of mini-batches.

Yet, if we carry out a line search to determine the step size at some iteration k, we are basing our choice on the behavior of the stochastic loss f_k (depending in practice on the randomly drawn samples in mini-batch) along direction d_k. The term $x^k - x^{k-1}$ being added to the stochastic gradient, on the other hand, comes from a variables update that led a decrease in f_{k-1}; if f_k and f_{k-1} substantially differ from each other, the momentum direction $x^k - x^{k-1}$ might be unrelated to the new mini-batch loss, and possibly even point to increasing values of f_k. It might thus be not so unusual for the value of β to be shrunk to very small values to obtain the decent property, and then several backtracks might also be needed to obtain sufficiently small step sizes guaranteeing the sufficient decrease. In order to overcome this issue, we would need f_{k-1} to be somewhat similar to f_k.

3.2 Leveraging mini-batch persistency

For the problem highlighted in the previous Sect. 3.1, we propose a practical solution that can be employed when the stochastic functions f_k and g_k are of the form (2).

The strategy to enforce some form of similarity between f_k and f_{k-1} consists in the selection of some indexes from the preceding mini-batch B_{k-1} when the new mini-batch B_k is formed, so that

$$\mathcal{R}_{k-1} = B_{k-1} \cap B_k \neq \emptyset.$$

In the context of machine learning, this idea translates into using a shared portion of data samples in successive mini-batches. The larger the overlap between mini-batches, the more similar f_{k-1} and f_k will be. This strategy is referred to as *mini-batch persistency*, and it was observed to be beneficial even with vanilla SGD algorithms, especially when large batch sizes are considered [26].

This idea will be crucial for the rest of the work. We therefore believe it is important to provide here readers with some additional arguments, of numerical nature. We thus present the results of preliminary computational experiments aimed at assessing the effect of mini-batch persistency in mini-batch GD optimization. We considered two learning tasks:

Table 1 Total number of times, during training, that the momentum term is not a descent direction for a possible new mini-batch with various overlap percentages

Problem	Batch size	Overlap				
		0%	25%	50%	75%	100%
ijcnn	128	5692	32	19	12	10
	512	1552	111	51	29	11
MNIST	128	54,319	28,282	10,345	2069	5
	512	14,745	6533	1325	64	0

- a convex problem—an RBF-kernel classifier to be trained on the ijcnn dataset[1] from LIBSVM [37];
- a nonconvex problem—a multi-layer perceptron with 1000 hidden units and ReLU activations to be trained on the MNIST [38] dataset.[2]

Both problems have been solved by means of vanilla mini-batch GD: for the two problems we set a fixed learning rate of 10^3 and 10^{-2}, respectively.

During the execution of the algorithm, at the beginning of each iteration we measure the angle between the momentum term $x^k - x^{k-1}$ and the (negative) stochastic gradient $-\nabla f_k(x^k)$ that we would get using different mini-batches B_k obtained for various levels of overlap. In particular, we consider mini-batches with 0%, 25%, 50%, 75% and 100% overlap (the latter showing what would happen if optimization were kept going on the same mini-batch). Note that the actual update will always be done with no momentum and no mini-batch overlap.

The test was repeated for batch sizes 128 and 512, to observe the effect of persistency with small and large batch sizes.

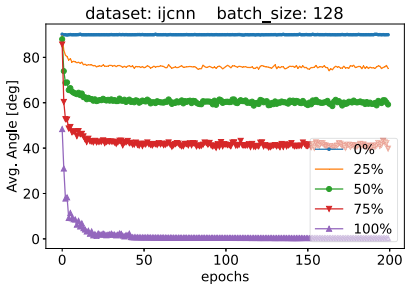
Figure 1 shows the results of this analysis. We can clearly observe that the greater is the overlap, the closer the momentum direction is to $-\nabla f_k(x^k)$. We thus deduce that, unsurprisingly, with overlapped mini-batches functions f_{k-1} and f_k end up being more similar; this way, larger values of β will be valid to obtain an overall heavy-ball direction of descent for f_k. In Table 1 we further report additional information on the overall number of times $x^k - x^{k-1}$ resulted in a non-descent direction for f_k at x^k; the conclusions we can draw are similar as above.

We conclude the preliminary discussion on mini-batch persistency remarking that, in practice, any mini-batch GD type optimizer can benefit from overlapped mini-batches.

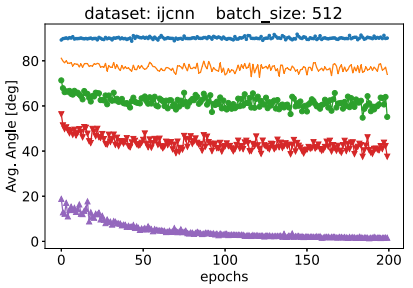
A convenient solution to force a 50% overlap in random reshuffling implementations [39] of mini-batch GD is to partition training data into $2N_b$ disjoint sets $\mathcal{R}_1^k, \ldots, \mathcal{R}_{2N_b}^k$ at each epoch κ and work, at iteration j within the epoch, with the subset $B_j^\kappa = \mathcal{R}_{j-1}^\kappa \cup \mathcal{R}_j^\kappa$, so that, for all $j = 2, \ldots, 2N_b$, we have $B_j^\kappa \cap \mathcal{B}_{j+1}^\kappa = \mathcal{R}_j$ and $B_j^\kappa \cap B_h^\kappa = \emptyset$ for all $h > j + 1$. With this approach, each data point is used twice (in subsequent iterations) within an epoch and the number of updates per epoch is doubled. Yet, given the sequential usage of the partitions $\mathcal{R}_j$, we can keep data in RAM/VRAM without the

[1] Available at csie.ntu.edu.tw/ cjlin/libsvmtools/datasets.

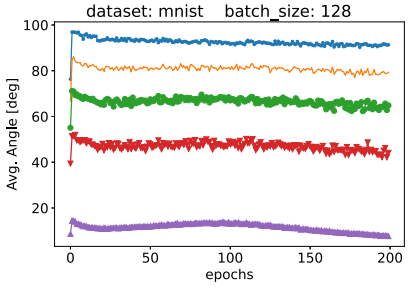
[2] Available at yann.lecun.com/exdb/mnist/.



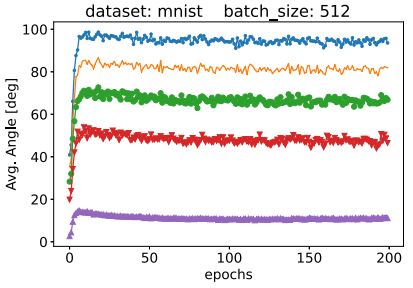
(a) Kernel classifier on `ijcnn` - batch size 128.



(b) Kernel classifier on `ijcnn` - batch size 512.



(c) Neural network classifier on `MNIST` - batch size 128.



(d) Neural network classifier on `MNIST` - batch size 512.

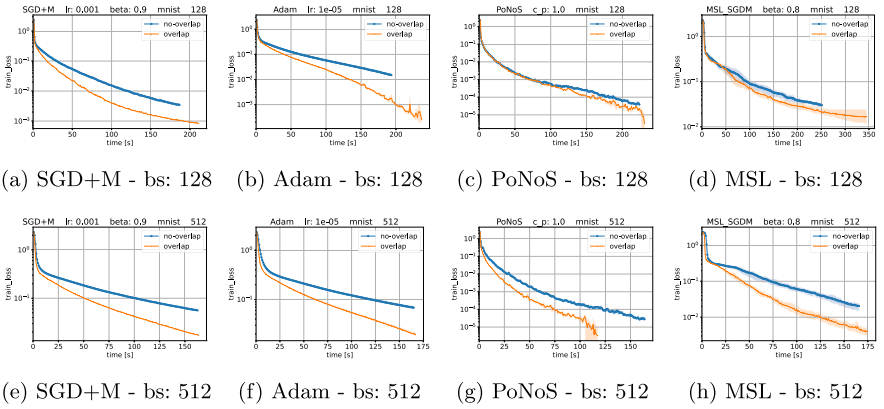
Fig. 1 Average angle per mini-batch GD epoch (in degrees) between the momentum term and the negative stochastic gradient we would obtain for a new mini-batch with 0%, 25%, 50%, 75% and 100% persistency

need of reading the same data from disk twice per epoch. In other words, when overlap is used, the I/O time from disk required per epoch (which is one of the dominating components in the computational cost of algorithms in large-scale learning) does not increase w.r.t. the non-overlap case.

At this point, we find it useful to show how a 50% mini-batch persistency affects SGD with momentum, Adam [8], PoNoS [18] and MSL_SGDM [25]. To this aim, we look at the progress of the training loss through time on the `MNIST` problem already employed above. Each algorithm is run—for 3 different random seeds—for a maximum of 200 epochs and is stopped earlier in case a solution x^k s.t. $f(x^k) < 10^{-5}$ is found. Batch sizes of 128 and 512 are tested. The results shown in Fig. 2 suggest that a 50% overlap can lead to a significant improvement in the performance of these methods in the tasks considered.

4 Data-persistent conjugate gradient rules

The possibility of exploiting mini-batch persistency opens up a range of options to determine a suitable value for β_k in heavy-ball, so as to



(a) SGD+M - bs: 128 (b) Adam - bs: 128 (c) PoNoS - bs: 128 (d) MSL - bs: 128

(e) SGD+M - bs: 512 (f) Adam - bs: 512 (g) PoNoS - bs: 512 (h) MSL - bs: 512

Fig. 2 Effect of a 50% mini-batch persistency with state-of-the-art algorithms (Minibatch-GD with momentum, Adam, PoNoS, MSL-SGDM) applied to the problem of training a multi-layer perceptron on the `MNIST` dataset. Time is specified in seconds

(i) obtain a descent direction for f_k at x^k;
(ii) get a large enough momentum term, not to shoot down its nice contribution for the overall optimization process.

The particular strategy we propose in this paper stems from the strong connection existing between heavy-ball and nonlinear conjugate gradient (CG) methods. Indeed, in both classes of algorithms, the new search direction is a linear combination of the current gradients and the previous direction.

The conceptual distinction between heavy-ball and CG is subtle and not unambiguously recognized by the optimization community. There is a substantial equivalence in the convex quadratic case, if heavy-ball parameters are chosen optimally [34]. In the general nonlinear case, CG methods define the search direction by means of a predetermined formula for β_k and then carry out a Wolfe line search; on the other hand, heavy-ball selects α_k and β_k altogether by some rule. If an Armijo-type line search is employed after choosing α_k and β_k according to some rule, as done, e.g., in [25, 29], we are arguably somewhere in a middle ground. We propose a framework that belongs to this middle ground in the stochastic setting.

Remark 1 In the recent literature, two conjugate-gradient methods for finite-sum problem with convergence guarantees have been proposed [40, 41]. The two algorithms are substantially direct descendants of classical CG methods in nonlinear optimization, both possessing convergence properties in expectation. However, they present two major differences w.r.t. what we present here, which actually make their employment very unappealing in deep learning scenarios: they require a Wolfe-type line search, significantly increasing the per-iteration cost, and they employ variance reduction techniques, as the algorithm presented in [40] requires periodic full batch evaluations of the gradient, and the one presented in [41] stores per sample gradients, which have a massive impact on the memory footprint of the method.

In the general case, conjugate gradient methods perform a momentum update along a direction given by

$$d_k = -\nabla f(x^k) + \beta_k d_{k-1} = -\nabla f(x^k) + \frac{\beta_k}{\alpha_{k-1}}(x^k - x^{k-1}).$$

Several formulae have been proposed that can be used to compute β_k, that collapse to the same rule when f is quadratic [34]. Given $y_k = \nabla f(x^k) - \nabla f(x^{k-1})$, some well-known formulations for computing β_k (cfr. [27, Sect. 12.5]) are:

$$\begin{aligned}
\beta_k^{HS} &= \frac{\nabla f(x^k)^T y_k}{(x^k - x^{k-1})^T y_k} && \text{(Hestenes–Stiefel)},\\
\beta_k^{FR} &= \frac{||\nabla f(x^k)||^2}{||\nabla f(x^{k-1})||^2} && \text{(Fletcher–Reeves)},\\
\beta_k^{PPR} &= \frac{\nabla f(x^k)^T y_k}{||\nabla f(x^{k-1})||^2} && \text{(Polyak–Polak–Ribiére)}.
\end{aligned} \tag{11}$$

We shall note that, with the above rules, we exploit the information available during iteration $k-1$ to define the coefficient β_k for iteration k. Thus, in a stochastic scenario, rules of this kind would only be reasonable if f_k did not change too much from an iteration to the other. This condition somewhat becomes true under mini-batch persistency.

Indeed, let $B_k \cap B_{k+1} = \mathcal{R}_k \neq \emptyset$ and

$$f_{\mathcal{R}_k} = \frac{1}{|\mathcal{R}_k|} \sum_{i \in \mathcal{R}_k} f_i(x).$$

We can use the values of $\nabla f_{\mathcal{R}_k}(x^k)$ and $\nabla f_{\mathcal{R}_k}(x^{k+1})$, that are available during iteration k, to compute the value β_{k+1} according to one of the above formulae. For instance, Fletcher-Reeves update becomes

$$\beta_{k+1} = \frac{\|\nabla f_{\mathcal{R}_k}(x^{k+1})\|^2}{\|\nabla f_{\mathcal{R}_k}(x^k)\|^2}.$$

The momentum term at iteration $k+1$ will thus be suitably selected for the function $f_{\mathcal{R}_k}$, which actually constitutes a substantial portion of the function f_{k+1} to be optimized at iteration $k+1$.

The advantage of this strategy is twofold: not only the estimated β_k will be meaningful when the mini-batch changes, but we also do not need to make additional evaluations, since the four quantities involved in the CG-type formulae would be computed anyway throughout the algorithm. Note that using $\mathcal{R}_k$ to compute β_{k+1} substantially differs from the strategy used in [40, 41], as this allows to compute the CG update rule exactly for the function $f_{\mathcal{R}_k}$.

5 The algorithmic framework

It is clear that the strategy presented in Sect. 4, although reasonable, has a heuristic nature, since β_k is estimated without looking at a portion (say a half) of currently considered data. To obtain a sound algorithm, we must ensure that the obtained direction is actually of descent for f_k, before starting a line search procedure along the direction: if that were not the case, the backtracking algorithm could get stuck in an infinite loop.

Clearly, if $x^k - x^{k-1}$ is a direction of descent for f_k, i.e., $(x^k - x^{k-1})^T \nabla f_k(x^k) < 0$, the descent property will hold for d_k for any choice of $\beta_k \geq 0$. From what we observed in the numerical results in Sect. 3.2, this will often be the case in practice. Yet, in the unfortunate case the direction d obtained by the CG-type formula does not satisfy the above conditions, we shall resort to any of the following recovery strategies:

- switch to the negative stochastic gradient $-g_k(x^k) = -\nabla f_k(x^k)$ as a search direction;
- invert the search direction and take $d_k = -d$;
- employ the damping strategy from [25];
- compute the direction according to the strategy proposed in [29], i.e.,

$$d_k = -a g_k(x^k) + b(x^k - x^{k-1}),$$
$$(a, b) \in \arg\min_{p \in \mathbb{R}^2} \frac{1}{2} p^T H_k p + p^T \begin{bmatrix} \|g_k(x^k)\|^2 \\ g_k(x^k)^T (x^k - x^{k-1}) \end{bmatrix},$$

  where the sequence of matrices $\{H_k\} \subseteq \mathbb{R}^{2\times 2}$ satisfies suitable uniform boundedness assumptions.

The complete pseudocode of the resulting proposed method is reported in Algorithm 1. A few aspects are worth to be remarked about the algorithm:

(1) In practical implementations, the algorithm will often be designed to work in a random-reshuffling fashion [39]. However, for the sake of simplicity and generality, we describe it in a "pure" stochastic setup.
(2) At the first iteration, a pure SGD step is performed.
(3) The set $\mathcal{R}_k$ identifies the training samples that are reused in consecutive mini-batches, i.e. $B_k \cap B_{k+1} = \mathcal{R}_k$. In practice, at line 17 of the algorithm, we can build $\mathcal{R}_k$ selecting from B_k the indices that were drawn in the most recent iterations at step 5, so that during an epoch we avoid using the same training samples for many iterations in a row. For example, when using a 50% overlap, we will have $\mathcal{R}_k = \mathcal{S}_k$ so as to use each training sample twice per epoch. As we highlight in Sect. 5.1, using a deterministic rule at line 17 is crucial to characterize the bias of the gradient estimate $g_k(x^k)$.
(4) We do not specify the particular way of choosing the tentative step size α_0^k at line 14. There are various options that might work both theoretically and computationally. Of course, a fixed value could be used for all iterations; however, we might prefer to define it adaptively. For this latter approach, a heuristic method, such as the one proposed by Vaswani et al. [17], could be considered, or we could use the generalized Stochastic Polyak step size introduced by Wang et al. [24].

(5) Finally, we shall underline that Algorithm 1 is shown to be using a monotone Armijo-type line search at step 15. This is done to ease the description of the method, but nothing prohibits to employ a nonmonotone line search.

Algorithm 1 Mini-Batch Conjugate Gradient with Data Persistency (MBCG-DP)

1: Input: $x^0 \in \mathbb{R}^n$, $f_1, \ldots, f_N : \mathbb{R}^n \to \mathbb{R}$, $\gamma \in (0,1)$, $\delta \in (0,1)$.
2: Let $\mathcal{R}_0 = \emptyset$,
3: $\beta_1 = 0$, $d_0 = 0$
4: **for** $k = 1, 2, \ldots$ **do**
5: Randomly draw a subset $\mathcal{S}_k \subset \{1, \ldots, N\} \setminus \mathcal{R}_{k-1}$
6: Set $B_k = \mathcal{R}_{k-1} \cup \mathcal{S}_k$
7: Let $f_k = \frac{1}{|B_k|} \sum_{i \in B_k} f_i$ and $g_k = \frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i$
8: Set $d = -g_k(x^k) + \beta_k d_{k-1}$
9: **if** $d^T g_k(x^k) < 0$ **then**
10: Set $d_k = d$
11: **else**
12: Define d_k such that $d_k^T g_k(x^k) < 0$
13: **end if**
14: Define a suitable initial step size α_0^k
15: Set $\alpha_k = \max_{j=0,1,\ldots} \{\alpha_0^k \delta^j \mid f_k(x^k + \alpha_0^k \delta^j d_k) \leq f_k(x^k) + \gamma \alpha_0^k \delta^j g_k(x^k)^T d_k\}$
16: Set $x^{k+1} = x^k + \alpha_k d_k$
17: Select $\mathcal{R}_k \subset B_k$ according to a deterministic rule
18: Define β_{k+1} from $\nabla f_{\mathcal{R}_k}(x^k)$ and $\nabla f_{\mathcal{R}_k}(x^{k+1})$ according to any CG rule (11)
19: **end for**

5.1 Convergence analysis

We now turn to the presentation of a formal convergence analysis for Algorithm 1. The analysis tackles two main issues, that are addressed in two separate parts.

5.1.1 Estimator bias in presence of mini-batch persistency

The key element of most convergence analyses for stochastic methods, including the one we carry out in the next subsection, relies on the availability of a conditionally unbiased estimator $g_k(x)$ of $\nabla f(x^k)$, i.e. $\mathbb{E}_k[g_k(x^k)] = \nabla f(x^k)$. We use $\mathbb{E}_k[\cdot]$ to denote the conditional expectation with respect to x^k, meaning $\mathbb{E}_k[\cdot] = \mathbb{E}[\cdot \mid x^k]$, while $\mathbb{E}[\cdot]$ indicates the total expectation.

However, in contrast to classical setups, where the unbiased estimator is reasonably obtained by drawing at each iteration the mini-batch to be used completely at random, we employ a data persistency strategy that turns this assumption false, as the set of indexes $\mathcal{R}_{k-1}$ is fixed, and thus deterministic, for the iteration k.

Let us denote $\bar{\mathcal{S}} = \{1, \ldots, N\} \setminus \mathcal{S}$. Recalling that $\nabla f_{\mathcal{S}}(x) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(x)$, and assuming for simplicity that $\mathcal{S}_k$ is randomly drawn from $\bar{\mathcal{R}}_{k-1}$ (so that $\mathbb{E}_k[\nabla f_{\mathcal{S}_k}(x^k)] =$

$\nabla f_{\bar{\mathcal{R}}_{k-1}}(x^k)$) we get that

$$\begin{aligned}\mathbb{E}_k[g_k(x^k)] &= \mathbb{E}_k\left[\frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)\right]\\ &= \mathbb{E}_k\left[\frac{1}{|B_k|}\left(\sum_{i\in\mathcal{R}_{k-1}}\nabla f_i(x^k)+\sum_{i\in\mathcal{S}_k}\nabla f_i(x^k)\right)\right]\\ &= \frac{1}{|B_k|}\sum_{i\in\mathcal{R}_{k-1}}\nabla f_i(x^k)+\mathbb{E}_k\left[\frac{1}{|B_k|}\sum_{i\in\mathcal{S}_k}\nabla f_i(x^k)\right]\\ &= \frac{|\mathcal{R}_{k-1}|}{|B_k|}\nabla f_{\mathcal{R}_{k-1}}(x^k)+\frac{|\mathcal{S}_k|}{|B_k|}\mathbb{E}_k\left[\nabla f_{\mathcal{S}_k}(x^k)\right]\\ &= \frac{|\mathcal{R}_{k-1}|}{|B_k|}\nabla f_{\mathcal{R}_{k-1}}(x^k)+\frac{|\mathcal{S}_k|}{|B_k|}\nabla f_{\bar{\mathcal{R}}_{k-1}}(x^k).\end{aligned} \tag{12}$$

If, for instance, we assume $|\mathcal{R}_{k-1}| = |\mathcal{S}_k| = |B_k|/2$, we get

$$\mathbb{E}_k[g_k(x^k)] = \frac{1}{2}\nabla f_{\mathcal{R}_{k-1}}(x^k)+\frac{1}{2}\nabla f_{\bar{\mathcal{R}}_{k-1}}(x^k),$$

which is a gradient estimate that is clearly biased towards the "persistent" samples. When taking the expected value for the gradient estimator, the weight assigned to the terms $\nabla f_i(x^k)$ for $i \in \mathcal{R}_{k-1}$ is $\frac{1}{2|\mathcal{R}_{k-1}|}$, evidently higher than it would be in the common unbiased stochastic gradient, i.e., $\frac{1}{N}$.

Fortunately, we can recover unbiasedness of the estimator applying a suitable correction when constructing the approximation $g_k(x^k)$—and similarly $f_k(x^k)$. This result is formalized hereafter.

Proposition 5.1 *Let $\mathcal{R}_{k-1} \subset \{1,\dots,N\}$ and let $\mathcal{S}_k$ be a random subset of $\bar{\mathcal{R}}_{k-1} = \{1,\dots,N\} \setminus \mathcal{R}_{k-1}$. Let $\zeta_k = \frac{N-|\mathcal{R}_{k-1}|}{|\mathcal{S}_k|}$. Then, the quantities*

$$f_k(x^k) = \frac{1}{N}\left(\sum_{i\in\mathcal{R}_{k-1}} f_i(x^k)+\zeta_k\sum_{i\in\mathcal{S}_k} f_i(x^k)\right)$$

and

$$g_k(x^k) = \frac{1}{N}\left(\sum_{i\in\mathcal{R}_{k-1}}\nabla f_i(x^k)+\zeta_k\sum_{i\in\mathcal{S}_k}\nabla f_i(x^k)\right)$$

are conditionally unbiased estimators of $f(x^k)$ and $\nabla f(x^k)$ given x^k, i.e., $\mathbb{E}_k[f_k(x^k)] = f(x^k)$ and $\mathbb{E}_k[g_k(x^k)] = \nabla f(x^k)$.

Proof We only prove the result for g_k, as the proof for f_k is substantially identical. Taking the conditional expected value of $g_k(x^k)$ given x^k, recalling that $\frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}}\nabla f_i(x) = \nabla f_{\mathcal{S}}(x)$, that $\mathbb{E}_k[\nabla f_{\mathcal{S}_k}(x^k)] = \nabla f_{\bar{\mathcal{R}}_{k-1}}(x^k)$ and (12), we get

$$\begin{aligned}
\mathbb{E}_k[g_k(x^k)] &= \mathbb{E}_k\left[\frac{1}{N}\left(\sum_{i\in\mathcal{R}_{k-1}}\nabla f_i(x^k) + \zeta_k\sum_{i\in\mathcal{S}_k}\nabla f_i(x^k)\right)\right] \\
&= \frac{|\mathcal{R}_{k-1}|}{N}\nabla f_{\mathcal{R}_{k-1}}(x^k) + \zeta_k\frac{|\mathcal{S}_k|}{N}\nabla f_{\bar{\mathcal{R}}_{k-1}}(x^k) \\
&= \frac{1}{N}\left(\sum_{i\in\mathcal{R}_{k-1}}\nabla f_i(x^k) + \frac{N-|\mathcal{R}_{k-1}|}{|\mathcal{S}_k|}\frac{|\mathcal{S}_k|}{N-|\mathcal{R}_{k-1}|}\sum_{i\in\bar{\mathcal{R}}_{k-1}}\nabla f_i(x^k)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(x^k) = \nabla f(x^k),
\end{aligned}$$

thus getting the thesis. □

Remark 2 The correction to make the gradient estimate unbiased even in presence of mini-batch persistency allows to soundly employ the latter technique from a theoretical convergence standpoint. However, we note that the correction heavily alleviates the sought effects that drove the introduction of the overlap strategy itself in the first place, since the weight assigned to the "persistent" samples is drastically lowered. This is also evident in the computational experiments, as shown later in Fig. 3. Therefore, we believe the theoretical analysis of convergence with a biased estimator would also be of great interest. Yet, this issue is a large open problem and its treatment would go far beyond the scope of this work; we thus leave it to future studies.

5.1.2 Properties of the search direction

Together with the assumption on gradient estimate unbiasedness, largely discussed in the previous subsection, we need to state another important assumption for properly analyzing the algorithm: the search directions d_k shall in fact be somehow tied to the true gradient $\nabla f(x^k)$. As pointed out in well-established literature (see, e.g., [4, Sect. 4.1]), the sequence of search directions $\{d_k\}$ shall be such that, for all k,

$$\mathbb{E}_k[\|d_k\|] \leq C_1\|\nabla f(x^k)\|, \quad \text{and} \quad \mathbb{E}_k[d_k]^T\nabla f(x^k) \leq -C_2\|\nabla f(x^k)\|^2.$$

Following the discussion in [30], this can be ensured if, given constants c_1, c_2 and c_3, the entire sequence $\{d_k\}$ satisfies the following set of conditions:

$$\|d_k\| \leq c_1\|g_k(x^k)\|, \tag{13}$$

$$d_k^T g_k(x^k) \leq -c_2\|g_k(x^k)\|^2, \tag{14}$$

$$\mathrm{Var}_k(d^k) \leq c_3\,\mathrm{Var}_k(g_k(x^k)), \tag{15}$$

where by Var_k we denote the conditional variance of a vector, i.e., $\mathrm{Var}_k(v) = \mathbb{E}_k[\|v\|^2] - \|\mathbb{E}_k[v]\|^2$.

Properties (13)–(14) can be checked, as safeguards, at each iteration for fixed values of c_1 and c_2: when not satisfied by the CG type direction, the algorithm can switch to the stochastic gradient direction.

In order to ensure that condition (15) is also satisfied by the direction resulting from the safeguard condition, we need to keep parameter β_k bounded; this is also not particularly restrictive, as it can be enforced by a clipping operation at the update time; in other words, we can simply set $\beta_{k+1} = \min\{\hat{\beta}_{k+1}, \bar{\beta}\}$, where $\hat{\beta}_{k+1}$ is the value resulting from the employed CG update rule. With this precaution, we only have to rely on an assumption on the variance of β_{k+1}, which appears reasonable as argued in [30, Remark 2].

The satisfaction of condition (15) for the whole sequence, if clipping and safeguards are suitably applied, is formally stated and proved in the next proposition.

Proposition 5.2 *Let, for all k, $d_k = -g_k(x^k) + \beta_k d_{k-1}$, where*

$$\beta_k = \begin{cases} \min\{\hat{\beta}_k, \bar{\beta}\} & \text{if } \|p_k\| \leq c_1 \|g_k(x^k)\| \text{ and } p_k^T g_k(x^k) \leq -c_2 \|g_k(x^k)\|^2, \\ 0 & \text{otherwise}, \end{cases}$$

with $p_k = -g_k(x^k) + \min\{\hat{\beta}_k, \bar{\beta}\} d_{k-1}$. Assuming that, for all k, $\|d_{k-1}\| \leq M$ and $\mathrm{Var}_k(\beta_k) \leq \hat{c}_3 \mathrm{Var}_k(g_k(x^k))$ for some constants $M, \hat{c}_3 > 0$, we have that $\mathrm{Var}_k(d^k) \leq c_3 \mathrm{Var}_k(g_k(x^k))$, with $c_3 = \left(1 + M^2 \hat{c}_3 + 2M\sqrt{\hat{c}_3}\right)$.

Proof The variance of the direction d_k is given by

$$\begin{aligned} \mathrm{Var}_k(-g_k(x^k) + \beta_k d_{k-1}) = \mathrm{Var}_k(g_k(x^k)) + \|d_{k-1}\|^2 \mathrm{Var}_k(\beta_k) \\ - 2\,\mathrm{Cov}_k(g_k(x^k), \beta_k d_{k-1}). \end{aligned}$$

Exploiting the Cauchy–Schwartz inequality on the dot product $u^T v$ where $u_i = \sqrt{\mathrm{Var}_k((g_k(x^k))_i)}$ and $v_i = |(d_{k-1})_i|$, we can write

$$\begin{aligned} |\,\mathrm{Cov}_k(g_k(x^k), \beta_k d_{k-1})| &\leq \sum_{i=1}^{n} \left|\mathrm{Cov}_k((g_k(x^k))_i, \beta_k (d_{k-1})_i)\right| \\ &\leq \sum_{i=1}^{n} \sqrt{\mathrm{Var}_k((g_k(x^k))_i)} \sqrt{\mathrm{Var}_k(\beta_k (d_{k-1})_i)} \\ &= \sqrt{\mathrm{Var}_k(\beta_k)} \sum_{i=1}^{n} \sqrt{\mathrm{Var}_k((g_k(x^k))_i)} |(d_{k-1})_i| \\ &\leq \sqrt{\mathrm{Var}_k(\beta_k)} \sqrt{\mathrm{Var}_k(g_k(x^k))} \|d_{k-1}\|. \end{aligned}$$

Using that $\|d_{k-1}\| \leq M$ and that $\mathrm{Var}_k(\beta_k) \leq \hat{c}_3 \mathrm{Var}_k(g_k(x^k))$ we obtain

$$\begin{aligned}\mathrm{Var}_k(d_k) &\leq \mathrm{Var}_k(g_k(x^k)) + \|d_{k-1}\|^2 \mathrm{Var}_k(\beta_k) + 2\sqrt{\mathrm{Var}_k(\beta_k)}\sqrt{\mathrm{Var}_k(g_k(x^k))}\|d_{k-1}\| \\ &\leq \left(1 + M^2\hat{c}_3 + 2M\sqrt{\hat{c}_3}\right) \mathrm{Var}_k(g_k(x^k)),\end{aligned}$$

which concludes the proof. □

Proposition 5.2 provides the final building block for establishing convergence results for the proposed algorithm. Note that the first assumption on $\|d_{k-1}\|$ is not restrictive, as it can be enforced at iteration k by simple clipping of the then deterministic direction d_{k-1} (i.e., using $\min\left\{1, \frac{M}{\|d_{k-1}\|}\right\} d_{k-1}$ instead of d_{k-1}).

5.1.3 Convergence results

We are now ready to present the asymptotic convergence analysis for Algorithm 1. In particular, we are showing how the method proposed in this paper possesses analogous properties as the other well-known stochastic line-search based methods from the literature [16–18, 30, 42]. Specifically, it is possible to guarantee fast convergence of the algorithmic framework under interpolation and PL assumptions [13, 34, 43].

We recall that the minimizer interpolation property [43, Definition 3] states that

$$x^* \in \arg\min_{x^* \in \mathbb{R}^n} f(x) \implies x^* \in \arg\min_{x^* \in \mathbb{R}^n} f_i(x) \quad \text{for all } i \in \{1, \ldots, N\}.$$

On the other hand, we say that a function f satisfies the PL condition if

$$\exists \mu > 0 \quad \text{such that } 2\mu(f(x) - f^*) \leq ||\nabla f(x)||^2 \quad \text{for all } x \in \mathbb{R}^n.$$

We shall also recall that (see, e.g., [30, Sect. 3]), L-smoothness, minimizers interpolation and the PL condition imply the *Strong Growth Condition* to hold for some $\rho > 0$, i.e.,

$$\mathbb{E}_k[\|\nabla f_k(x)\|^2] \leq \rho \|\nabla f(x)\|^2 \quad \text{for all } x \in \mathbb{R}^n \text{ and for any } k.$$

Since we need to insert into the algorithmic framework the fixes described in Sects. 5.1.1 and 5.1.2, we report for the sake of clarity the instructions of the provably convergent version of the algorithm in Algorithm 2. Note that we also made explicit the constant lower bound required for the initial step size $\alpha_k^0 \geq \bar{\alpha}$ of the line search procedure.

The main convergence theorem is stated hereafter. The result is recovered by [30, Theorem 5.3].

Theorem 5.3 *Assume $f : \mathbb{R}^n \to \mathbb{R}$ is an L-smooth function, satisfying the PL condition with constant μ and the minimizer interpolation property, and that the randomly drawn functions f_k considered at any iteration k are L_k-smooth functions, with $L \leq L_{max} = \max_k L_k$. Let the sequence $\{\beta_k\}$ be such that $\mathrm{Var}_k(\beta_k) \leq \hat{c}_3 \mathrm{Var}_k(g_k(x^k))$*

Algorithm 2 Convergent MBCG−DP

Input: $x^0 \in \mathbb{R}^n$, $f_1, \ldots, f_N : \mathbb{R}^n \to \mathbb{R}$, $\gamma \in (0,1)$, $\delta \in (0,1)$, $c_1, c_2, \bar{\alpha}, \bar{\beta}, M > 0$.
Let $\mathcal{R}_0 = \emptyset$,
$\beta_1 = 0$, $d_0 = 0$
for $k = 1, 2, \ldots$ **do**
  Randomly draw a subset $\mathcal{S}_k \subset \{1, \ldots, N\} \setminus \mathcal{R}_{k-1}$
  Set $B_k = \mathcal{R}_{k-1} \cup \mathcal{S}_k$
  Set $\zeta_k = \frac{N - |\mathcal{R}_{k-1}|}{|\mathcal{S}_k|}$
  Let $f_k = \frac{1}{N}\left(\sum_{i \in \mathcal{R}_{k-1}} f_i + \zeta_k \sum_{i \in \mathcal{S}_k} f_i\right)$
  Let $g_k = \frac{1}{N}\left(\sum_{i \in \mathcal{R}_{k-1}} \nabla f_i + \zeta_k \sum_{i \in \mathcal{S}_k} \nabla f_i\right)$
  Set $d = -g_k(x^k) + \beta_k \min\left\{1, \frac{M}{\|d_{k-1}\|}\right\} d_{k-1}$
  if $\|d\| \leq c_1 \|g_k(x^k)\|$ **and** $d^T g_k(x^k) \leq -c_2 \|g_k(x^k)\|^2$ **then**
    Set $d_k = d$
  else
    Set $d_k = -g_k(x^k)$
  end if
  Define an initial step size $\alpha_0^k \geq \bar{\alpha}$
  Set $\alpha_k = \max_{j=0,1,\ldots}\{\alpha_0^k \delta^j \mid f_k(x^k + \alpha_0^k \delta^j d_k) \leq f_k(x^k) + \gamma \alpha_0^k \delta^j g_k(x^k)^T d_k\}$
  Set $x^{k+1} = x^k + \alpha_k d_k$
  Select $\mathcal{R}_k \subset B_k$ according to a deterministic rule
  Define $\hat{\beta}_{k+1}$ from $\nabla f_{\mathcal{R}_k}(x^k)$ and $\nabla f_{\mathcal{R}_k}(x^{k+1})$ according to any CG rule (11)
  Set $\beta_{k+1} = \min\{\hat{\beta}_{k+1}, \bar{\beta}\}$
end for

and that $c_2 > c_3\left(1 - \frac{1}{\rho}\right)$, *being* $c_3 = \left(1 + M^2\hat{c}_3 + 2M\sqrt{\hat{c}_3}\right)$ *and* ρ *the SGC constant. Then, if* $\{x^k\}$ *is the sequence produced by Algorithm 2 the following property holds*:

$$\mathbb{E}[f(x^{k+1}) - f(x^*)] \leq (\eta\bar{\alpha})^k (f(x^0) - f(x^*)), \tag{16}$$

where $\eta = \left(\frac{L_{max} c_1^2}{2c_2}\left(\frac{1}{\gamma} + \frac{1}{\delta(1-\gamma)}\right) - 2\left(c_2 - c_3\left(1 - \frac{1}{\rho}\right)\right)\mu\right)$.
Consequently, when all the constants involved are such that $0 < \eta < \frac{1}{\bar{\alpha}}$, *the convergence rate is linear with an* $\mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$ *iteration, stochastic function evaluations and stochastic gradient evaluations complexity to obtain an* ϵ*-accurate solution in expectation.*

Proof The sequence of directions $\{d_k\}$ satisfies the conditions (13)–(14) with constants c_1 and c_2 as a result of steps 11–14. By Proposition 5.2 and the instructions of the algorithm, it also satisfies condition (15) with c_3 defined as in the statement of the theorem. All the assumptions of [30, Theorem 5.3] are thus satisfied by the ones here, and we consequently get the thesis. □

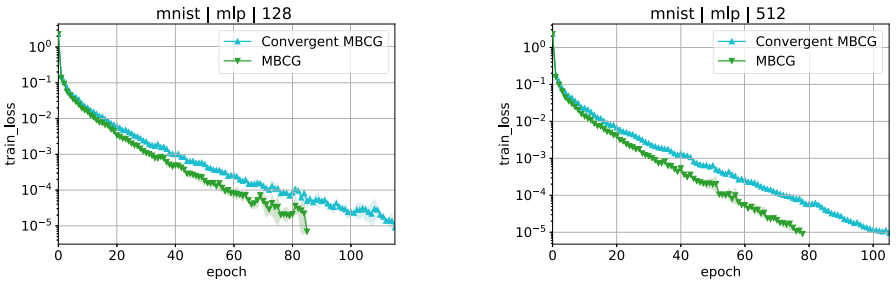


Fig. 3 Behavior of MBCG-DP and its unbiased variant when employed to train a multi-layer perceptron on `MNIST` dataset with batch size of 128 and 512

6 Computational experiments

In this section, we present the results of a comprehensive experimental analysis of the proposed Mini-Batch Conjugate Gradient method with Data Persistency (MBCG-DP). As previously discussed in Sect. 5.1, the bias correction term required to make the gradient estimate unbiased actually opposes to the effect we introduce with the overlap strategy (for completeness, we show in Fig. 3 that bias correction indeed slows down the algorithm on the `MNIST` task considered in Sect. 3.2). In the following computational experiments we therefore consider the algorithm version without any bias correction. All experiments reported in this section were implemented in `Python3`[3] and run on a machine with the following characteristics: Ubuntu 22.04, Intel i5-13400F Processor, 32 GB RAM, Nvidia GeForce RTX 4060 8GB.

The initial analysis focuses on the behavior of MBCG-DP when using different configurations. Specifically, we aim at identifying

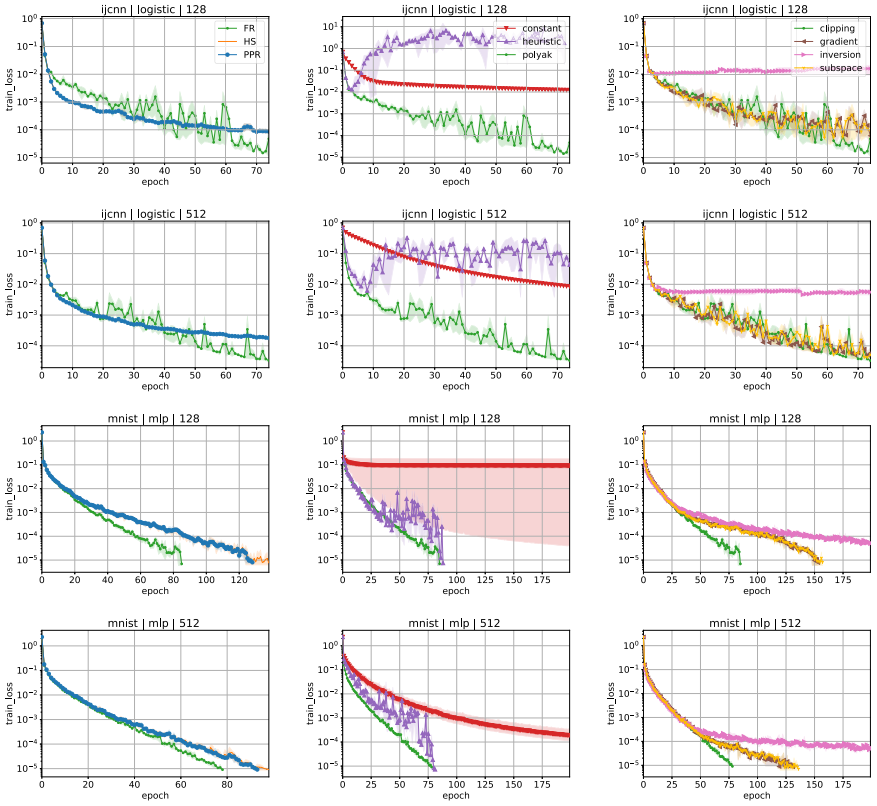
(a) the best CG rule to select β_k,
(b) the optimal strategy for selecting the tentative step size α_0^k in the line-search,
(c) the most effective method for modifying d_k when it is not a descent direction for the current mini-batch objective.

We consider in this phase the same two learning problems as in Sect. 3.2: an RBF-kernel classifier on the `ijcnn` dataset and a multi-layer perceptron on the `MNIST` dataset. The algorithm is run in a random-reshuffling fashion, with 50% mini-batch persistency, and batch sizes of 128 and 512 are considered for both tasks. Standard parameters are employed for the nonmonotone Armijo-type line search, with $\gamma = 0.5$, $\delta = 0.5$. For each experiment 3 different seeds are considered.

In Fig. 4a we reported the behavior of MBCG-DP when comparing the CG rules of Fletcher–Reeves (FR), Polyak–Polak–Ribiére (PPR) and Hestenes–Stiefel (HS). In this experiment the generalized SPS (7) with $c = 1$ is used to select α_0^k and the momentum clipping approach (contraction factor 0.5) from [25] is employed in case d_k is not a descent direction. The Fletcher–Reeves rule clearly resulted the most effective one in both tasks and for both batch sizes.

In Fig. 4b we show the behavior of MBCG-DP when using different approaches for selecting the tentative step α_0^k for the line search: a constant value, the heuris-

[3] The code for the proposed method is available at github.com/dadoPuccio/MB-Conjugate-Gradient-DP.



(a) Comparison of CG rules. (b) Comparison of tentative step sizes α_0^k. (c) Comparison of the recovery strategies for d_k.

Fig. 4 Behavior of MBCG-DP when employed to train a kernel classifier on `ijcnn` and a multi-layer perceptron on `MNIST` dataset. The left column refers to the comparison of the FR, HS and PPR rules for selecting β_k; the central column refers to the comparison of rules for selecting α_0^k (constant, heuristic, SPS); the right column refers to different safeguard strategies for ensuring a descent direction is used (clipping, gradient, inversion, subspace)

tic employed in [17, 25] and the generalized SPS (7) with $c = 1$ were tested. The Fletcher–Reeves rule is used in this experiment; in case d_k is not a descent direction the repeated momentum clipping approach is used, again with contraction factor of 0.5. The generalized SPS proved to be the most effective choice.

Finally, in Fig. 4c we observe the impact of different recovery options to retrieve a descent direction, namely: switching to the simple (negative) stochastic gradient direction, recomputing the direction according to the subspace optimization strategy proposed in [29], inverting the direction and taking $-d_k$ and adopting the repeated momentum clipping approach (contraction factor of 0.5) from [25]. In this experiment, the Fletcher–Reeves rule is employed in combination with the generalized SPS initial step size. The clipping strategy resulted the preferable choice especially in nonconvex problems.

This detailed comparison allows us to identify a promising setup to run MBCG-DP in further experiments to compare it to state-of-the-art methods. In what follows, we thus employed the Fletcher–Reeves rule for β_k, the generalized stochastic Polyak step size (7) for choosing α_0^k and the clipping strategy on β_k as a recovery option for obtaining a descent direction. The resulting method will be denoted by the acronym MBCG_FR.

The other algorithms included in the following experiments are: vanilla SGD with momentum, Adam [8], SLS [17], PoNoS [18] and MSL_SGDM [25]. The benchmark of test problems include three convex instances, consisting in the training of an RBF-kernel classifier on `mushrooms`, `ijcnn` and `rcv1` datasets[4] Moreover, three nonconvex learning tasks are taken into account: once again a multi-layer perceptron with 1000 hidden units and ReLU activations to be trained on `MNIST`, a three-layered convolutional architecture with 32 channels each, followed by a fully-connected layer of 512 units to be trained on `FashionMNIST` [44] dataset[5] and a ResNet18 architecture to be trained on `CIFAR10` [45] dataset[6]. For the latter network, we employed the normalization-free setup proposed in [46]. Batch sizes of 128 and 512 were again considered for all problems.

In order to offer a fair comparison between the proposed MBCG-DP method and its competitors, each reference algorithm was evaluated to assess whether employing mini-batch persistency could enhance its performance. For each problem considered, the reference algorithms are tested using an overlap of 0%, 10%, 25%, 50% and 75%. An algorithm is deemed to perform better with a certain overlap value if it either terminates before the others with a superior solution or, at the termination time of the first configuration to finish, it holds a better solution. The training loss is compared to asses whether a solution is better than the other. We report in Table 2 the outcome of this preliminary analysis. In convex problems, we observe that overlap is almost always beneficial. However, with larger architectures in nonconvex tasks, this benefit does not appear to hold. We argue that, in such cases, the dominant computational burden shifts from data I/O operations to the evaluation of the training loss and its gradient. Since the number of iterations per epoch with a 50% overlap is doubled, in this cases the execution time per epoch will also double. In addition, we remark that hyperparameters were selected for each considered method based on preliminary experiments carried out without employing data persistency. A specialized selection of hyperparameters values for each method and overlap percentage might further enhance performance when data persistency is employed.

We report in Fig. 5 and in Fig. 6 the comparison of MBCG_FR to its competitors in terms of the training loss and the validation accuracy. The algorithms are compared in terms of wall-clock execution time, since employing mini-batch overlap changes the number of updates per epoch, making epoch-based comparison unfair. The averages of 3 runs with different random seeds are reported. Standard parameters are employed for the monotone line-search of SLS and MSL_SGDM, with $\gamma = 0.1$ and $\delta = 0.5$, and also for the nonmonotone line search of PoNoS and MBCG_FR, with $\gamma = 0.5$ and

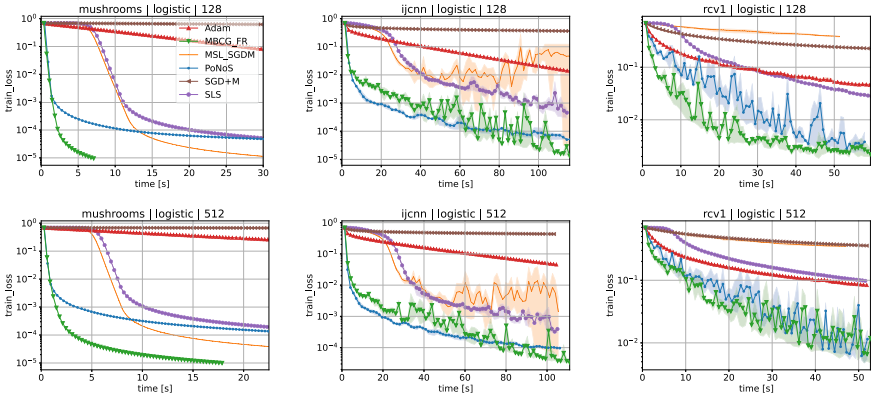
[4] datasets available at csie.ntu.edu.tw/ cjlin/libsvmtools/datasets..

[5] available at github.com/zalandoresearch/fashion-mnist.

[6] available at cs.toronto.edu/ kriz/cifar.html.

Table 2 Mini-batch overlap settings leading to the best-performing algorithms across problems

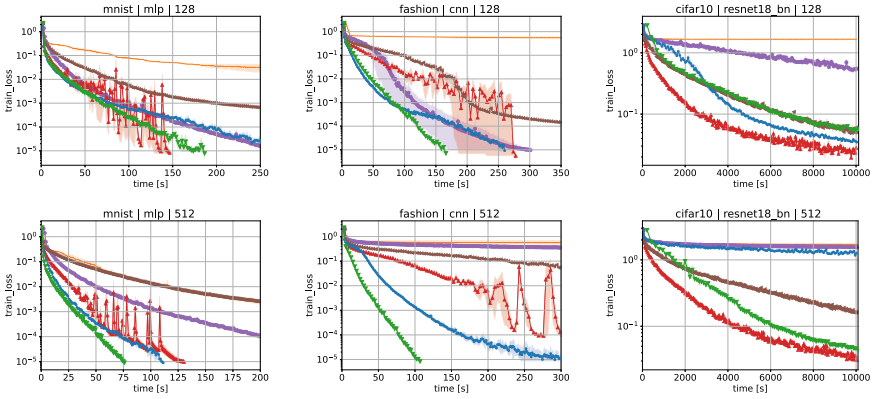
	RBF-kernel classifier mushrooms		ijcnn		rcv1		MLP MNIST		CNN fashionMNIST		ResNet18 CIFAR10	
	128 (%)	512 (%)	128 (%)	512 (%)	128 (%)	512 (%)	128 (%)	512 (%)	128 (%)	512 (%)	128 (%)	512 (%)
SGD+M	75	75	75	75	75	75	75	75	75	75	0	0
Adam	75	75	75	75	75	75	50	50	75	75	0	10
SLS	75	75	50	75	75	75	0	75	0	0	0	0
PoNoS	75	75	75	75	50	75	25	50	0	0	0	0
MSL_SGDM	75	75	50	75	0	25	10	10	75	0	25	0



(a) RBF-kernel classifier on `mushrooms` dataset.

(b) RBF-kernel classifier on `ijcnn` dataset.

(c) RBF-kernel classifier on `rcv1` dataset.



(d) Multi-layer perceptron on `MNIST` dataset.

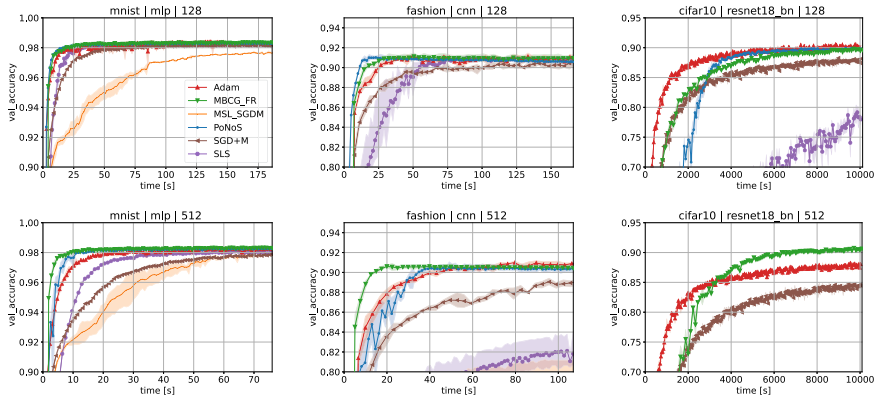
(e) CNN on `FashionMNIST` dataset.

(f) ResNet18 on `CIFAR10` dataset.

Fig. 5 Comparison of the proposed MBCG_FR algorithm with other algorithms in the literature. The training loss over time is shown for each problem

$\delta = 0.5$. For SGD with momentum we use $\alpha = 10^{-2}$, $\beta = 0.9$ for convex problems and $\alpha = 10^{-3}$, $\beta = 0.9$ for nonconvex ones. For Adam we use $\alpha = 10^{-3}$ for convex problems and $\alpha = 10^{-4}$ for nonconvex ones. For both PoNoS and MBCG_FR we use $c = 1$ in the definition of the generalized SPS, except for the problem of training the ResNet18 architecture on `CIFAR10` dataset, where $c = 0.2$ is employed. Reference algorithms utilize mini-batch persistency only in configurations specified in Table 2 with the values there reported, whereas MBCG_FR always uses a 50% overlap.

As shown in Fig. 5, MBCG_FR is extremely effective at reducing the training loss. In all convex tasks it is capable of obtaining high-quality solutions more quickly than the other algorithms considered. In general, MBCG_FR is also competitive in nonconvex problems, especially for a batch size of 512. Moreover, looking at trend of the validation accuracy reported in Fig. 6, we can note that even in cases where Adam is



(a) Multi-layer perceptron on `MNIST` dataset.

(b) CNN on `FashionMNIST` dataset.

(c) ResNet18 on `CIFAR10` dataset.

Fig. 6 Comparison of the proposed MBCG_FR algorithm with other algorithms in the literature. The validation accuracy over time is shown for each problem

slightly faster, MBCG_FR is capable of reaching top out-of-sample prediction results. In addition, examining Fig. 6c, we can note that the best overall validation accuracy value is reached on the `CIFAR10` problem by MBCG_FR with a batch size of 512.

Based on the results reported in this section, we conclude that the proposed MBCG-DP algorithm is capable of achieving state-of-the-art performance in both convex and nonconvex finite-sum problems, making it particularly suitable for large-scale tasks and machine learning applications. This is especially true when considerable computing resources are available, allowing larger batch sizes to be employed.

7 Conclusions

In this work, we introduced an algorithmic framework based on stochastic line searches and momentum directions to tackle nonlinear finite-sum optimization problems in deep learning scenarios. In order to truly benefit from the addition of momentum terms in the search direction, the method leverages the concept of mini-batch persistency. The resulting approach, for which we state formal convergence results under suitable assumptions, proves to be computationally efficient and effective at solving the class of problems under consideration. In addition, our empirical analysis shows that, for simpler models or small architectures, any algorithm could benefit from the usage of mini-batch persistency.

There are two main research directions that shall be followed at this point: from the one hand, a deep investigation of the convergence properties of algorithms that employ mini-batch persistency in absence of bias-correction terms would be of significant impact. On the other hand, further computational experiments on even larger models, such as transformer architectures, would be important to assess the usefulness of the proposed approach in the most modern applications.

Acknowledgements The authors are grateful to the Editor and the reviewers of this manuscript for their constructive comments that helped us to significantly improve the quality of this work. We are very thankful to Professors M. Sciandrone, S. Lucidi and G. Liuzzi for the useful discussions in the early stages of this research.

Funding Open access funding provided by Università degli Studi di Firenze within the CRUI-CARE Agreement. No funding was received for conducting this study.

Data Availability Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Code Availability The code developed for the experimental part of this paper is publicly available at github.com/dadoPuccio/MB-Conjugate-Gradient-DP.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**(3), 400–407 (1951). https://doi.org/10.1214/aoms/1177729586
2. Polyak, B.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**(5), 1–17 (1964). https://doi.org/10.1016/0041-5553(64)90137-5
3. Nesterov, Y.: A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. Proc. USSR Acad. Sci. **269**, 543–547 (1983)
4. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. **60**(2), 223–311 (2018)
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(61), 2121–2159 (2011)
6. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
7. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701 (2012). arxiv:1212.5701
8. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015 (2015). arxiv:1412.6980
9. Schmidt, M., Le Roux, N., Bach, F.: Minimizing finite sums with the stochastic average gradient. Math. Program. **162**(1), 83–112 (2017). https://doi.org/10.1007/s10107-016-1030-6
10. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: Burges, C.J., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, 2013, vol. 26. Curran Associates, Inc., Lake Tahoe (2013). https://proceedings.neurips.cc/paper_files/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Paper.pdf

11. Gower, R.M., Kunstner, F., Schmidt, M.: Variance reduced model based methods: new rates and adaptive step sizes. In: OPT 2023: Optimization for Machine Learning, 2023 (2023). https://openreview.net/forum?id=7Ie3aoqJg1
12. Defazio, A., Bottou, L.: On the ineffectiveness of variance reduced optimization for deep learning. In: Advances in Neural Information Processing Systems, 2019, vol. 32. Curran Associates, Inc., New York (2019). https://proceedings.neurips.cc/paper_files/paper/2019/file/84d2004bf28a2095230e8e14993d398d-Paper.pdf
13. Ma, S., Bassily, R., Belkin, M.: The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In: International Conference on Machine Learning, 2018, pp. 3325–3334. PMLR (2018)
14. Vaswani, S., Kunstner, F., Laradji, I.H., Meng, S.Y., Schmidt, M., Lacoste-Julien, S.: Adaptive gradient methods converge faster with over-parameterization (and you can do a line-search). CoRR abs/2006.06835 (2020). arXiv:2006.06835
15. Mutschler, M., Zell, A.: Parabolic approximation line search for DNNs. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, 2020, vol. 33, pp. 5405–5416. Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/3a30be93eb45566a90f4e95ee72a089a-Paper.pdf
16. Loizou, N., Vaswani, S., Hadj Laradji, I., Lacoste-Julien, S.: Stochastic Polyak step-size for SGD: an adaptive learning rate for fast convergence. In: Proceedings of The 24th International Conference on Artificial Intelligence and Statistics, 2021, vol. 130, pp. 1306–1314. PMLR (2021). https://proceedings.mlr.press/v130/loizou21a.html
17. Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., Lacoste-Julien, S.: Painless stochastic gradient: interpolation, line-search, and convergence rates. In: Advances in Neural Information Processing Systems, 2019, vol. 32. Curran Associates, Inc., Vancouver Convention Center, Vancouver (2019). https://proceedings.neurips.cc/paper_files/paper/2019/file/2557911c1bf75c2b643afb4ecbfc8ec2-Paper.pdf
18. Galli, L., Rauhut, H., Schmidt, M.: Don't be so monotone: relaxing stochastic line search in over-parameterized models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, 2023, vol. 36, pp. 34752–34764. Curran Associates, Inc., Ernest N. Morial Convention Center, New Orleans (2023). https://proceedings.neurips.cc/paper_files/paper/2023/file/6d0bf1265ea9635fb4f9d56f16d7efb2-Paper-Conference.pdf
19. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning, 2013, pp. 1139–1147. PMLR (2013)
20. Sebbouh, O., Gower, R.M., Defazio, A.: Almost sure convergence rates for stochastic gradient descent and stochastic heavy ball. In: Conference on Learning Theory, 2021, pp. 3935–3971. PMLR (2021)
21. Tseng, P.: An incremental gradient (-projection) method with momentum term and adaptive stepsize rule. SIAM J. Optim. **8**(2), 506–531 (1998)
22. Jelassi, S., Li, Y.: Towards understanding how momentum improves generalization in deep learning. In: International Conference on Machine Learning, 2022, pp. 9965–10040. PMLR (2022)
23. Gitman, I., Lang, H., Zhang, P., Xiao, L.: Understanding the role of momentum in stochastic gradient methods. In: Advances in Neural Information Processing Systems, 2019, vol. 32 (2019)
24. Wang, X., Johansson, M., Zhang, T.: Generalized Polyak step size for first order optimization with momentum. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, 2023, vol. 202, pp. 35836–35863. PMLR, Ernest N. Morial Convention Center, New Orleans (2023). https://proceedings.mlr.press/v202/wang23l.html
25. Fan, C., Vaswani, S., Thrampoulidis, C., Schmidt, M.: MSL: an adaptive momentum-based stochastic line-search framework. In: OPT 2023: Optimization for Machine Learning, 2023 (2023). https://openreview.net/forum?id=UfvQbl7Kpx
26. Fischetti, M., Mandatelli, I., Salvagnin, D.: Faster SGD training by minibatch persistency (2018). arXiv:1806.07353 [cs.LG]
27. Grippo, L., Sciandrone, M.: Introduction to Methods for Nonlinear Optimization, vol. 152. Springer, Cham (2023)
28. Chan-Renous-Legoubin, R., Royer, C.W.: A nonlinear conjugate gradient method with complexity guarantees and its application to nonconvex regression. EURO J. Comput. Optim. **10**, 100044 (2022)

29. Lapucci, M., Liuzzi, G., Lucidi, S., Sciandrone, M.: A globally convergent gradient method with momentum. arXiv preprint (2024). arXiv:2403.17613
30. Lapucci, M., Pucci, D.: Convergence conditions for stochastic line search based optimization of over-parametrized models. Optimization (2025). https://doi.org/10.1080/02331934.2025.2551250
31. Ghadimi, S., Lan, G.: Stochastic first- and zeroth-order methods for nonconvex stochastic programming. SIAM J. Optim. **23**(4), 2341–2368 (2013)
32. Liang, T., Rakhlin, A.: Just interpolate: kernel "Ridgeless" regression can generalize. Ann. Stat. **48**(3), 1329–1347 (2020). https://doi.org/10.1214/19-AOS1849
33. Liu, C., Zhu, L., Belkin, M.: Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. Appl. Comput. Harmon. Anal. **59**(Special Issue on Harmonic Analysis and Machine Learning), 85–116 (2022). https://doi.org/10.1016/j.acha.2021.12.009
34. Polyak, B.T.: Introduction to Optimization, vol. 1, p. 32. Optimization Software, Inc., Publications Division, New York (1987)
35. Armijo, L.: Minimization of functions having Lipschitz continuous first partial derivatives. Pac. J. Math. **16**(1), 1–3 (1966)
36. Wright, S.J., Recht, B.: Optimization for Data Analysis. Cambridge University Press, Cambridge (2022)
37. Chang, C.-C., Lin, C.-J.: LibSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. (2011). https://doi.org/10.1145/1961189.1961199
38. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
39. Gürbüzbalaban, M., Ozdaglar, A., Parrilo, P.A.: Why random reshuffling beats stochastic gradient descent. Math. Program. **186**, 49–84 (2021)
40. Jin, X.-B., Zhang, X.-Y., Huang, K., Geng, G.-G.: Stochastic conjugate gradient algorithm with variance reduction. IEEE Trans. Neural Netw. Learn. Syst. **30**(5), 1360–1369 (2019). https://doi.org/10.1109/TNNLS.2018.2868835
41. Kou, C., Yang, H.: A mini-batch stochastic conjugate gradient algorithm with variance reduction. J. Glob. Optim. **87**(2), 1009–1025 (2023)
42. Vaswani, S., Bach, F., 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, 2019, pp. 1195–1204. PMLR (2019)
43. Mishkin, A.: Interpolation, growth conditions, and stochastic gradient descent. PhD Thesis, University of British Columbia (2020)
44. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. https://doi.org/10.48550/arXiv.1708.07747
45. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images: Technical Report. University of Toronto, Toronto (2009)
46. Civitelli, E., Sortino, A., Lapucci, M., Bagattini, F., Galvan, G.: A robust initialization of residual blocks for effective ResNet training without batch normalization. IEEE Trans. Neural Netw. Learn. Syst. (2023). https://doi.org/10.1109/TNNLS.2023.3325541