A Subsampling Line-Search Method with Second-Order Results

E. Bergou * Y. Diouane† V. Kunc ‡ V. Kungurtsev § C. W. Royer ¶

March 23, 2020

Abstract

In many contemporary optimization problems such as those arising in machine learning, it can be computationally challenging or even infeasible to evaluate an entire function or its derivatives. This motivates the use of stochastic algorithms that sample problem data, which can jeopardize the guarantees obtained through classical globalization techniques in optimization such as a trust region or a line search. Using subsampled function values is particularly challenging for the latter strategy, which relies upon multiple evaluations. On top of that all, there has been an increasing interest for nonconvex formulations of data-related problems, such as training deep learning models. For such instances, one aims at developing methods that converge to second-order stationary points quickly, i.e., escape saddle points efficiently. This is particularly delicate to ensure when one only accesses subsampled approximations of the objective and its derivatives.

In this paper, we describe a stochastic algorithm based on negative curvature and Newton-type directions that are computed for a subsampling model of the objective. A line-search technique is used to enforce suitable decrease for this model, and for a sufficiently large sample, a similar amount of reduction holds for the true objective. By using probabilistic reasoning, we can then obtain worst-case complexity guarantees for our framework, leading us to discuss appropriate notions of stationarity in a subsampling context. Our analysis encompasses the deterministic regime, and allows us to identify sampling requirements for second-order line-search paradigms. As we illustrate through real data experiments, these worst-case estimates need not be satisfied for our method to be competitive with first-order strategies in practice.

Keywords: Nonconvex optimization, finite-sum problems, subsampling methods, negative curvature, worst-case complexity.

*MaIAGE, INRAE, Université Paris-Saclay, 78350 Jouy-en-Josas, France (elhoucine.bergou@inra.fr). King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. This author received support from the AgreenSkills+ fellowship programme which has received funding from the EU's Seventh Framework Programme under grant agreement No FP7-609398 (AgreenSkills+ contract).

†ISAE-SUPAERO, Université de Toulouse, 31055 Toulouse Cedex 4, France (youssef.diouane@isae.fr).

‡Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague (kuncvlad@fel.cvut.cz). Support for this author was provided by the CTU SGS grant no. SGS17/189/OHK3/3T/13.

§Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague (vyacheslav.kungurtsev@fel.cvut.cz). Support for this author was provided by the OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

¶LAMSADE, CNRS, Université Paris-Dauphine, Université PSL, 75016 PARIS, FRANCE (clement.royer@dauphine.psl.eu). Support for this author was partially provided by Subcontract 3F-30222 from Argonne National Laboratory.

1 Introduction

In this paper, we aim to solve

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

where the objective function f is not necessarily convex and the components f_i are assumed to be twice-continuously differentiable on $\mathbb{R}^n$. We are interested in problems in which the number of components $N \geq 1$ is extremely large: for this reason, we will assume that it is computationally infeasible to evaluate the entire function, its gradient or its Hessian.

To overcome this issue, we consider the use of *subsampling techniques* to compute stochastic estimates of the objective function, its gradient and its Hessian. Given a random set $\mathcal{S}$ sampled from $\{1,\dots,N\}$ and a point $x \in \mathbb{R}^n$, we will use

$$m(x;\mathcal{S}) := \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} f_i(x), \quad g(x;\mathcal{S}) := \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \nabla f_i(x), \quad \text{and} \quad H(x;\mathcal{S}) := \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \nabla^2 f_i(x), \tag{2}$$

to estimate the quantities $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$, respectively, where $|\mathcal{S}|$ denotes the cardinal number of the sampling set $\mathcal{S}$. We are interested in iterative minimization processes that use a different, randomly selected sampling set $\mathcal{S}$ at every iteration.

The standard subsampling optimization procedure for such a problem is the (batch) stochastic gradient descent (SGD) method, wherein the gradient is estimated by a sample of the gradients $\{\nabla f_i\}$ and a step is taken in the negative of this direction: although the original SGD method relies on a single sample per iteration, using a batch of samples can be beneficial as it reduces the variance of the gradient estimate. The SGD framework is still quite popular in a variety of applications, including large-scale machine learning problems, particularly those arising from the training of deep neural net architectures [10]. However, it is known to be sensitive to nonconvexity, particularly in the context of training deep neural nets. For such problems, it has indeed been observed that the optimization landscape for the associated (nonconvex) objective exhibits a significant number of saddle points, around which the flatness of the function tends to slow down the convergence of SGD [19]. This behavior is typical of first-order methods, despite the fact that those schemes almost never converge to saddle points [33]. By incorporating second-order information, one can guarantee that saddle points can be escaped from at a favorable rate: various algorithms that provide such guarantees while only requiring gradient or Hessian-vector products have been proposed in the literature [1, 2, 34, 52]. Under certain accuracy conditions, which can be satisfied with arbitrarily high probability by controlling the size of the sample, these methods produce a sequence of iterates that converge to a local minimizer at a certain rate. Alternatively, one can extract second-order information and escape saddle points using accelerated gradient techniques in the stochastic setting [46]. The results are also in high probability, with a priori tuned step-sizes. Noise can be used to approximate second-order information as well [52]. Recent proposals [50, 53] derive high probability convergence results with second-order steps (e.g., Newton steps) based on sampled derivatives and exact objective values, by means of trust-region and cubic regularization frameworks.

Subsampling can be viewed as a particular case of stochastic optimization where the underlying distribution takes values within a discrete set. Stochastic subsampling Newton methods, including [4, 8, 12, 22, 39, 42, 51], present principled use of second-order information to accelerate the convergence while typically using a line search on exact function values to ensure

global convergence. In the general stochastic optimization setting, a variety of algorithms have been extended to handle access to (sub)sampled derivatives, and possibly function values: of particular interest to us are the algorithms endowed with complexity guarantees. When exact function values can be computed, one can employ strategies based on line search [15], cubic regularization [15, 27] or trust region [18, 24] to compute a step of suitable length. Many algorithms building on SGD require the tuning of the step size parameter (also called learning rate), which can be cumbersome without knowledge of the Lipschitz constant. On the contrary, methods that are based on a globalization technique (line search, trust region, quadratic or cubic regularization) can control the size of the step in an adaptive way, and are thus less sensitive to parameter tuning.

In spite of their attractive properties with respect to the step size, globalized techniques are challenging to extend to the context of inexact function values. Indeed, these methods traditionally accept new iterates only if they produce a sufficient reduction of the objective value. Nevertheless, inexact variants of these schemes have been a recent topic of interest in the literature. In the context of stochastic optimization, several trust-region algorithms that explicitly deal with computing stochastic estimates of the function values have been described [7, 17, 31]. In the specific case of least-squares problems, both approaches (exact and inexact function values) have been incorporated within a Levenberg-Marquardt framework [5, 6]. The use of stochastic function estimates in a line-search framework (a process that heavily relies on evaluating the function at tentative points) has also been the subject of very recent investigation. A study based on proprietary data [29] considered an inexact Newton and negative curvature procedure using each iteration's chosen mini-batch as the source of function evaluation sample in the line search. A more mature approach for line search was presented in [35], where extensive experiments matching performance to pre-tuned SGD were presented. An innovating technique based on a backtracking procedure for steps generated by a limited memory BFGS method for nonconvex problems using first-order information was recently proposed in [9], and first-order convergence results were derived. Finally, contemporary to the first version of this paper, a stochastic line-search framework was described by Paquette and Scheinberg [38]. Similarly to our scheme, this algorithm computes stochastic estimates for function and gradient values, which are then used within a line-search algorithm. However, the two methods differ in their inspiration and results: we provide more details about these differences in the next paragraph, and throughout the paper when relevant.

In this paper, we propose a line-search scheme with second-order guarantees based on subsampling function and derivative evaluations. Our method uses these subsampled values to compute Newton-type and negative curvature steps. Although our framework bears similarities with the approach of [38], the two algorithms are equipped with different analysis, each based on their own arguments from probability theory. The method of [38] is designed with first-order guarantees in mind (in particular, the use of negative curvature is not explored), and its complexity results are particularized to the nonconvex, convex and strongly convex cases; our work presents a line-search method that is dedicated to the nonconvex setting, and to the derivation of second-order results. As such, we are able to show a rate of convergence to points satisfying approximate second-order optimality conditions, in expectation. By contrast, earlier work on second-order guarantees for subsampling methods often focused on complexity bounds holding with a high probability (of accurate samples being taken at each iteration), disallowing poor outlier estimates of the problem function. We believe that our results form a complementary and interesting addition to the state-of-the-art literature on stochastic optimization methods.

We organize this paper as follows. In Section 2, we describe our proposed approach based on line-search techniques. In Section 3, we derive bounds on the amount of expected decrease that can be achieved at each iteration by our proposed approach. Section 4 gives the global convergence rate of our method under appropriate assumptions, followed by a discussion about the required properties and their satisfaction in practice. A numerical study of our approach is provided in Section 5. A discussion of conclusions and future research is given in Section 6.

Throughout the paper, $\|.\|$ denotes the Euclidean norm. A vector $v \in \mathbb{R}^n$ will be called a unit vector if $\|v\| = 1$. Finally, $\mathbb{I}_n$ denotes the identity matrix of size n.

2 Subsampling line-search method

In this section, we introduce a line-search algorithm dedicated to solving the unconstrained optimization problem (1): the detailed framework is provided in Algorithm 1.

At each iteration k, our method computes a random sampling set $\mathcal{S}_k$, and the associated estimates $g_k := g(x_k; \mathcal{S}_k)$ and $H_k := H(x_k; \mathcal{S}_k)$ of $\nabla f(x_k)$ and $\nabla^2 f(x_k)$, respectively. Since we have computed $\mathcal{S}_k$, the model $m_k(\cdot) := m(\cdot; \mathcal{S}_k)$ of the function f is also defined: the estimates $g_k := g(x_k; \mathcal{S}_k)$ and $H_k := H(x_k; \mathcal{S}_k)$ define the quadratic Taylor expansion of m_k around x_k, which we use to compute a search direction d_k. The form of this direction is set based on the values on the norm of g_k as well as the minimum eigenvalue of H_k, denoted by λ_k. The process is described through Steps 2-5 of Algorithm 1, and involves an optimality tolerance ϵ. When $\lambda_k < -\epsilon$, the Hessian estimate is indefinite, and we choose to use a negative curvature direction, as we know that it will offer sufficient reduction in the value of m_k (see the analysis of Section 3.2). When $\lambda_k \geq \|g_j\|^{1/2}$, the quadratic function defined by g_k and H_k is (sufficiently) positive definite, and this allows us to compute and use a Newton direction. Finally, when $\lambda_k \in \left[-\epsilon^{1/2}, \|g_k\|^{1/2}\right]$, we regularize this quadratic by an amount of order $\epsilon^{1/2}$, so that we fall back into the previous case: we thus compute a regularized Newton direction, for which we will obtain desirable decrease properties. Once the search direction has been determined, and regardless of its type, a backtracking line-search strategy is applied to select a step size α_k that decreases the model m_k by a sufficient amount (see condition (7)). This condition is instrumental in obtaining good complexity guarantees.

Aside from the use of subsampling, Algorithm 1 differs from the original method of Royer and Wright [44] in two major ways. First, we only consider three types of search direction, as opposed to five in the original method of Royer and Wright [44]. Indeed, in order to simplify the upcoming theoretical analysis, we do not allow for selecting gradient-based steps, i.e. steps that are colinear with the negative (subsampled) gradient. Note that such steps do not affect the complexity guarantees, but have a practical value since they are cheaper to compute. For this reason, we present the algorithm without their use, but we will re-introduce them in our numerical experiments. Secondly, our strategy for choosing among the three different forms for the direction differs slightly from Royer and Wright [44] in the first "if" condition on Steps 2 and 3 of the algorithm. As the results of Section 4 will show, in the fully sampled case, we will obtain results that are equivalent to that of the deterministic method [44] using this new "if" condition. Nevertheless, and for the reasons already mentioned in the first point, we will also revert to the original rule in our practical implementation (see Section 5.1).

Two comments about the description of Algorithm 1 are in order. First, we observe that the method as stated is not equipped with a stopping criterion. Apart from budget considerations,

Algorithm 1: A Line-search Algorithm based on Subsampling (ALAS).

Initialization: Choose $x_0 \in \mathbb{R}^n$, $\theta \in (0,1), \eta > 0$, $\epsilon > 0$.
for $k = 0, 1, \ldots$ **do**

1. Draw a random sample set $\mathcal{S}_k \subset \{1, \ldots, N\}$, and compute the associated quantities $g_k := g(x_k; \mathcal{S}_k), H_k := H(x_k; \mathcal{S}_k)$. Form the model m_k as a function of the variable s:

$$m_k(x_k + s) := m(x_k + s; \mathcal{S}_k). \tag{3}$$

2. Compute λ_k as the minimum eigenvalue of the Hessian estimate H_k.
If $\lambda_k \geq -\epsilon^{1/2}$ and $\|g_k\| = 0$ set $\alpha_k = 0,\ d_k = 0$ and go to Step 7.

3. If $\lambda_k < -\epsilon^{1/2}$, compute a negative eigenvector v_k such that

$$H_k v_k = \lambda_k v_k,\ \|v_k\| = -\lambda_k,\ v_k^\top g_k \leq 0, \tag{4}$$

set $d_k = v_k$ and go to the line-search step.

4. If $\lambda_k > \|g_k\|^{1/2}$, compute a Newton direction d_k solution of

$$H_k d_k = -g_k, \tag{5}$$

go to the line-search step.

5. If d_k has not yet been chosen, compute it as a regularized Newton direction, solution of

$$\left(H_k + (\|g_k\|^{1/2} + \epsilon^{1/2})\mathbb{I}_n\right) d_k = -g_k, \tag{6}$$

and go to the line-search step.

6. **Line-search step** Compute the minimum index j_k such that the step length $\alpha_k := \theta^{j_k}$ satisfies the decrease condition:

$$m_k(x_k + \alpha_k d_k) - m_k(x_k) \ \leq\ -\frac{\eta}{6}\alpha_k^3 \|d_k\|^3. \tag{7}$$

7. Set $x_{k+1} = x_k + \alpha_k d_k$.

8. Set $k = k + 1$.

end

one might be tempted to stop the method when the derivatives are suggesting that it is a second-order stationary point. However, since we only have access to subsampled versions of those derivatives, it is possible that we have not reached a stationary point for the true function. As we will establish later in the paper, one needs to take into account the accuracy of the model, and it might take several iterations to guarantee that we are indeed at a stationary point. We discuss the link between stopping criteria and stationarity conditions in Section 4.

Our second remark relates to the computation of a step. When the subsampled gradient g_k is zero and the subsampled matrix H_k is positive definite, we cannot compute a descent step using first- or second-order information, because the current iterate is second-order stationary for the subsampled model. In that situation, and for the reasons mentioned in the previous paragraph, we do not stop our method, but rather take a zero step and move on to a new iteration and a new sample set. As we will show in Section 4, after a certain number of such iterations, one can guarantee that a stationary point has been reached, with high probability.

3 Expected decrease guarantees with subsampling

In this section, we derive bounds on the amount of expected decrease at each iteration. When the current sample leads to good approximations of the objective and derivatives, we are able to guarantee decrease in the function for any possible step taken by Algorithm 1. By controlling the sample size, one can adjust the probability of having a sufficiently good model, so that the guaranteed decrease for good approximations will compensate a possible increase for bad approximations on average.

3.1 Preliminary assumptions and definitions

Throughout the paper, we will study Algorithm 1 under the following assumptions.

Assumption 3.1 *The function f is bounded below by $f_{\text{low}} \in \mathbb{R}$.*

Assumption 3.2 *The functions f_i are twice continuously differentiable, with Lipschitz continuous gradients and Hessians, of respective Lipschitz constants L_i and $L_{H,i}$.*

A consequence of Assumption 3.2 is that f is twice continuously differentiable, Lipschitz, with Lipschitz continuous first and second-order derivatives. This property also holds for $m(\cdot;\mathcal{S})$, regardless of the value of $\mathcal{S}$ (we say that the property holds *for all realizations of* $\mathcal{S}$). In what follows, we will always consider that $m(\cdot;\mathcal{S})$ and f have L-Lipschitz continuous gradients and L_H-Lipschitz continuous Hessians, where

$$L \;:=\; \max_i L_i, \qquad L_H \;:=\; \max_i L_{H,i}.$$

As a result of Assumption 3.2, there exists a finite positive constant U_H such that:

$$U_H \geq \max_{i=1,\ldots,N} \|\nabla^2 f_i(x_k)\| \quad \forall k.$$

The value $L = U_H$ is a valid one, however we make the distinction between the two by analogy with previous works [44, 50].

We will also require the following assumption.

Assumption 3.3 *There exists a finite positive constant U_g such that,*

$$U_g \geq \max_{i=1,\ldots,N} \|\nabla f_i(x_k)\| \quad \forall k,$$

for all realizations of the algorithm.

Assumption 3.3 is needed in order for the steps to be bounded in norm, which in turn guarantees that the possible increase of the function at a given iteration will also be bounded. Note that this assumption is less restrictive than assuming that all f_i's are Lipschitz continuous, or that there exists a compact set that contains the sequence of iterates. Because our sampling set is finite, the set of possibilities for each iterate is bounded (though it grows at a combinatorial pace). We thus believe that Assumption 3.3 is reasonable in our context, and it allows us to convey the main ideas behind our work. In practice, one could run the algorithm while detecting unboundedness of the iterates to potentially restart the method. In our numerical experiments we never had to resort to this case.

In the rest of the analysis, for every iteration k, we let

$$\pi_k := \frac{|\mathcal{S}_k|}{N}$$

denote the *sample fraction* used at every iteration. Our objective is to identify conditions on this fraction that allow for decrease in expectation.

Whenever the sample sets in Algorithm 1 are drawn at random, the subsampling process introduces randomness in an iterative fashion at every iteration. As a result, Algorithm 1 results in a stochastic process $\{x_k, d_k, \alpha_k, g_k, H_k, m_k(x_k), m_k(x_k+\alpha_k d_k)\}$ (we point out that the sample fractions need not be random). To lighten the notation throughout the paper, we will use these notations for the random variables and their realizations. Most of our analysis will be concerned with random variables, but we will explicitly mention that realizations are considered when needed. Our goal is to show that under certain conditions on the sequences $\{g_k\}$, $\{H_k\}$, $\{m_k(x_k)\}$, $\{m_k(x_k+\alpha_k d_k)\}$ the resulting stochastic process has desirable convergence properties in expectation.

Inspired by a number of definitions in the model-based literature for stochastic or subsampled methods [3, 17, 31, 34], we introduce a notion of sufficient accuracy for our model function and its derivatives.

Definition 3.1 *Given a realization of Algorithm 1 and an iteration index k, the model $m_k : \mathbb{R}^n \mapsto \mathbb{R}$ is said to be* $(\delta_f, \delta_g, \delta_H)$-accurate *with respect to (f, x_k, α_k, d_k) when*

$$|f(x_k) - m_k(x_k)| \leq \delta_f \quad \text{and} \quad |f(x_k + \alpha_k d_k) - m_k(x_k + \alpha_k d_k)| \leq \delta_f, \tag{8}$$

$$\|\nabla f(x_k) - g_k\| \leq \delta_g \quad \text{and} \quad \|\nabla f(x_k + \alpha_k d_k) - g(x_k + \alpha_k d_k, \mathcal{S}_k)\| \leq \delta_g, \tag{9}$$

as well as

$$\|\nabla^2 f(x_k) - H_k\| \leq \delta_H, \tag{10}$$

where $g_k := \nabla m_k(x_k)$, $H_k := \nabla^2 m_k(x_k)$ and $\delta_f, \delta_g, \delta_H$ are nonnegative constants.

Condition (8) is instrumental in establishing decrease guarantees for our method, while conditions (9) and (10) play a key role in defining proper notions of stationarity (see Section 4). Since we are operating with a sequence of random samples and models, we need a probabilistic equivalent of Definition 3.1, which is given below.

Definition 3.2 *Let $p \in (0, 1]$, $\delta_f \geq 0$, $\delta_g \geq 0$ and $\delta_H \geq 0$. A sequence of functions $\{m_k\}_k$ is called p-probabilistically $(\delta_f, \delta_g, \delta_H)$-accurate for Algorithm 1 if the events*

$$I_k := \{m_k \text{ is } (\delta_f, \delta_g, \delta_H)\text{-accurate with respect to } (f, x_k, \alpha_k, d_k)\}$$

satisfy

$$p_k := \mathbb{P}\left(I_k | \mathcal{F}_{k-1}\right) \geq p,$$

where $\mathcal{F}_{k-1}$ *is the* σ*-algebra generated by the sample sets* $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{k-1}$*, and we define* $\mathbb{P}(I_0|\mathcal{F}_{-1}) := \mathbb{P}(I_0)$*.*

Observe that if we sample the full data at every iteration (that is, $\mathcal{S}_k = \{1, \ldots, N\}$ for all k), the resulting model sequence satisfies the above definition for any $p \in [0, 1]$ and any positive values $\delta_f, \delta_g, \delta_H$. Given our choice of model (2), the accuracy properties are directly related to the random sampling set, but we will follow the existing literature on stochastic optimization and talk about accuracy of the models. We will however express conditions for good convergence behavior based on the sample size rather than the probability of accuracy.

In the rest of the paper, we assume that the estimate functions of the problem form a probabilistically accurate sequence as follows.

Assumption 3.4 *The sequence* $\{m_k\}_k$ *produced by Algorithm 1 is* p*-probabilistically* δ*-accurate, with* $\delta := (\delta_f, \delta_g, \delta_H)$ *and* $p \in (0, 1]$*.*

We now introduce the two notions of stationarity that will be considered in our analysis.

Definition 3.3 *Consider a realization of Algorithm 1, and let* ϵ_g, ϵ_H *be two positive tolerances. We say that the* k*-th iterate* x_k *is* (ϵ_g, ϵ_H)-model stationary *if*

$$\min\left\{\|g_k\|, \|g(x_{k+1}, \mathcal{S}_k)\|\right\} \leq \epsilon_g \quad \textit{and} \quad \lambda_k \geq -\epsilon_H. \tag{11}$$

Similarly, we will say that x_k *is* (ϵ_g, ϵ_H)-function stationary *if*

$$\min\left\{\|\nabla f(x_k)\|, \|\nabla f(x_{k+1})\|\right\} \leq \epsilon_g \quad \text{or} \quad \lambda_{\min}(\nabla^2 f(x_k)) \geq -\epsilon_H. \tag{12}$$

Note that the two definitions above are equivalent whenever the model consists of the full function, i.e. when $\mathcal{S}_k = \{1, \ldots, n\}$ for all k. We also observe that the definition of model stationarity involves the norm of the vector

$$g_k^+ := g(x_k + \alpha_k d_k; \mathcal{S}_k). \tag{13}$$

The norm of this "next gradient" is a major tool for the derivation of complexity results in Newton-type methods [14, 44]. In a subsampled setting, a distinction between g_k^+ and g_{k+1} is necessary because these two vectors are computed using different sample sets.

Our objective is to guarantee convergence towards a point satisfying a function stationarity property (12), yet we will only have control on achieving model stationarity. The accuracy of the models will be instrumental in relating the two properties, as shown by the lemma below.

Lemma 3.1 *Let Assumptions 3.1 and 3.2 hold. Consider a realization of the method that reaches an iterate* x_k *such that* x_k *is* (ϵ_g, ϵ_H)*-model stationary. Suppose further that the model* m_k *is* $(\delta_f, \delta_g, \delta_H)$*-accurate with*

$$\delta_g \leq \kappa_g \epsilon_g \quad \textit{and} \quad \delta_H \leq \kappa_H \epsilon_H \tag{14}$$

where κ_g *and* κ_H *are positive, deterministic constants independent of* k*. Then,* x_k *is a* $((1 + \kappa_g)\epsilon_g, (1 + \kappa_H)\epsilon_H)$*-function stationary point.*

Proof. Let x_k be an iterate such that

$$\min\{\|g_k\|, \|g_k^+\|\} \leq \epsilon_g \quad \text{and} \quad \lambda_k \geq -\epsilon_H.$$

Looking at the first property, suppose that $\|g_k\| \leq \epsilon_g$. In that case, we have:

$$\|\nabla f(x_k)\| \leq \|\nabla f(x_k) - g_k\| + \|g_k\| \leq \delta_g + \epsilon_g \leq (\kappa_g + 1)\epsilon_g.$$

A similar reasoning shows that if $\|g_k^+\| \leq \epsilon_g$, we obtain $\|\nabla f(x_{k+1})\| \leq (\kappa_g + 1)\epsilon_g$; thus, we must have

$$\min\{\|\nabla f(x_k)\|, \|\nabla f(x_{k+1})\|\} \leq (1 + \kappa_g)\epsilon_g.$$

Consider now a unit eigenvector v for $\nabla^2 f(x_k)$ associated with $\lambda_{\min}(\nabla^2 f(x_k))$, one has

$$\begin{aligned}\lambda_k - \lambda_{\min}(\nabla^2 f(x_k)) &\leq v^\top H_k v - v^\top \nabla^2 f(x_k) v \\ &\leq \|H_k - \nabla^2 f(x_k)\| \|v\|^2 \\ &\leq \delta_H.\end{aligned}$$

Hence, by using (14), one gets

$$\lambda_{\min}(\nabla^2 f(x_k)) = \big(\lambda_{\min}(\nabla^2 f(x_k)) - \lambda_k\big) + \lambda_k \geq -\delta_H - \epsilon_H \geq -(\kappa_H + 1)\epsilon_H.$$

Overall, we have shown that

$$\min\{\|\nabla f(x_k)\|, \|\nabla f(x_{k+1})\|\} \leq (1 + \kappa_g)\epsilon_g \quad \text{and} \quad \lambda_{\min}(\nabla^2 f(x_k)) \geq -(1 + \kappa_H)\epsilon_H,$$

and thus x_k is also a $((1 + \kappa_g)\epsilon_g, (1 + \kappa_H)\epsilon_H)$-function stationary point. ■

The reciprocal result of Lemma 3.1 will also be of interest to us in the next section.

Lemma 3.2 *Consider a realization of Algorithm 1 and the associated k-th iteration. Suppose that x_k is* not *$((1 + \kappa_g)\epsilon_g, (1 + \kappa_H)\epsilon_H)$-function stationary, and that the model m_k is $\delta = (\delta_f, \delta_g, \delta_H)$-accurate with $\delta_g \leq \kappa_g \epsilon_g$ and $\delta_H \leq \kappa_H \epsilon_H$ where κ_g and κ_H are positive constants. Then, x_k is* not *(ϵ_g, ϵ_H)-model stationary.*

3.2 A general expected decrease result

In this section, we study the guarantees that can be obtained (in expectation) for the various types of direction considered by our method. By doing so, we identify the necessary requirements on our sampling procedure, as well as on our accuracy threshold for the model values.

In what follows, we will make use of the following constants:

$$c_{nc} := \frac{3\theta}{L_H + \eta}, \; c_n := \min\left\{\left[\frac{2}{L_H}\right]^{1/2}, \left[\frac{3\theta}{L_H + \eta}\right]\right\}, \; c_{rn} := \min\left\{\frac{1}{1 + \sqrt{1 + L_H/2}}, \left[\frac{6\theta}{L_H + \eta}\right]\right\},$$

$$\bar{j}_{nc} := \left[\log_\theta\left(\frac{3}{L_H + \eta}\right)\right]_+, \; \bar{j}_n := \left[\log_\theta\left(\sqrt{\frac{3}{L_H + \eta}}\frac{\epsilon^{1/2}}{\sqrt{U_g}}\right)\right]_+, \; \bar{j}_{rn} := \left[\log_\theta\left(\frac{6}{L_H + \eta}\frac{\epsilon}{U_g}\right)\right]_+.$$

Those constants are related to the line-search steps that can be performed at every iteration of Algorithm 1. When the current iterate is not an approximate stationary point of the model, we can bound this number independently of k. This is the purpose of the following lemma.

Consider the event

$$E_k := \left\{ \min\{\|g_k\|, \|g_k^+\|\} > \epsilon \quad \text{or} \quad \lambda_k < -\epsilon^{1/2} \right\}, \tag{15}$$

where ϵ is the tolerance used in Algorithm 1.

Lemma 3.3 *Let Assumptions 3.1 and 3.2 hold for a realization of Algorithm 1. Consider an iteration k such that the event E_k holds. Then, the backtracking line search terminates with the step length $\alpha_k = \theta^{j_k}$, with $j_k \leq \bar{j} + 1$ and*

$$\alpha_k \|d_k\| \;\geq\; c\, \epsilon^{1/2}, \tag{16}$$

where

$$c := \min\{c_{nc}, c_n, c_{rn}\} \tag{17}$$

and

$$\bar{j} := \max\{\bar{j}_{nc}, \bar{j}_n, \bar{j}_{rn}\}. \tag{18}$$

Proof. We consider in turn the three possible steps that can be taken at iteration k, and obtain a lower bound on the amount $\alpha_k \|d_k\|$ for each of those.

Case 1: $\lambda_k < -\epsilon^{1/2}$ (negative curvature step). In that case, we apply the same reasoning than in [44, Proof of Lemma 1] with the model m_k playing the role of the objective, the backtracking line search terminates with the step length $\alpha_k = \theta^{j_k}$, with $j_k \leq \bar{j}_{nc} + 1$ and $\alpha_k \geq \frac{3\theta}{L_H + \eta}$. When d_k is computed as a negative curvature direction, one has $\|d_k\| = -\lambda_k > 0$. Hence,

$$\alpha_k \|d_k\| \;\geq\; \frac{3\theta}{L_H + \eta}[-\lambda_k] \;=\; c_{nc}[-\lambda_k] \geq c\epsilon^{1/2}.$$

Case 2: $\lambda_k > \|g_k\|^{1/2}$ (Newton step). Because (15) holds and $\lambda_k > 0$ in this case, we necessarily have $\|\tilde{g}_k\| = \min\{\|g_k\|, \|g_k^+\|\} > \epsilon$. From Algorithm 1, we know that d_k is chosen as the Newton step. Hence, using the argument of [44, Proof of Lemma 3] with $\epsilon_H = \epsilon^{1/2}$, the backtracking line search terminates with the step length $\alpha_k = \theta^{j_k}$, where

$$j_k \leq \left[\log_\theta \left(\sqrt{\frac{3}{L + \eta}} \frac{\epsilon^{1/2}}{\sqrt{U_g}} \right) \right]_+ + 1 = \bar{j}_n + 1,$$

thus the first part of the result holds. If the unit step size is chosen, we have by [44, Relation (23)] that

$$\alpha_k \|d_k\| = \|d_k\| \geq \left[\frac{2}{L_H} \right]^{1/2} \|g(x_k + \alpha_k d_k; \mathcal{S}_k)\|^{1/2} \geq c\epsilon^{1/2}. \tag{19}$$

Consider now the case $\alpha_k < 1$. Using [44, Relations (25) and (26), p. 1457], we have:

$$\|d_k\| \geq \frac{3}{L_H + \eta} \epsilon_H = \frac{3}{L_H + \eta} \epsilon^{1/2}$$

and

$$\alpha_k \geq \theta \sqrt{\frac{3}{L_H + \eta}} \epsilon_H^{1/2} \|d_k\|^{-1/2} = \theta \sqrt{\frac{3}{L_H + \eta}} \epsilon^{1/4} \|d_k\|^{-1/2}.$$

As a result,

$$\begin{aligned}\alpha_k\|d_k\| &\geq \theta\left[\frac{3}{L_H+\eta}\right]^{1/2}\epsilon^{1/4}\|d_k\|^{-1/2}\|d_k\| \\ \alpha_k\|d_k\| &\geq \left[\frac{3\theta}{L_H+\eta}\right]\epsilon^{1/2} \geq c\epsilon^{1/2}. \end{aligned} \tag{20}$$

Case 3: (Regularized Newton step) This case occurs when the conditions for the other two cases fail, that is, when $-\epsilon^{1/2} \leq \lambda_k \leq \|g_k\|^{1/2}$. We again exploit the fact that (15) holds to deduce that we necessarily have $\|\tilde{g}_k\| = \min\{\|g_k\|, \|g_k^+\|\} > \epsilon$. This in turn implies that $\min\{\|\tilde{g}_k\|\epsilon^{-1/2}, \epsilon^{1/2}\} \geq \epsilon^{1/2}$. As in the proof of the previous lemma, we apply the theory of [44, Proof of Lemma 4] using $\epsilon_H = \epsilon^{1/2}$. We then know that the backtracking line search terminates with the step length $\alpha_k = \theta^{j_k}$, with

$$j_k \leq \left[\log_\theta\left(\frac{6}{L_H+\eta}\frac{\epsilon}{U_g}\right)\right]_+ + 1 = \bar{j}_{rn} + 1,$$

since $\epsilon_H = \epsilon^{1/2}$.

We now distinguish between the cases $\alpha_k = 1$ and $\alpha_k < 1$. If the unit step size is chosen, we can use [44, relations 30 and 31], where $\nabla f(x_k + d_k)$ and ϵ_H are replaced by g_k^+ and $\epsilon^{1/2}$, respectively. This gives

$$\alpha_k\|d_k\| = \|d_k\| \geq \frac{1}{1+\sqrt{1+L_H/2}}\min\left\{\|g_k^+\|/\epsilon^{1/2}, \epsilon^{1/2}\right\}.$$

Therefore, if the unit step is accepted, one has by [44, equation 31]

$$\begin{aligned}\alpha_k\|d_k\| &\geq \frac{1}{1+\sqrt{1+L_H/2}}\min\left\{\|g_k^+\|\epsilon^{-1/2}, \epsilon^{1/2}\right\} \\ &\geq \frac{1}{1+\sqrt{1+L_H/2}}\min\left\{\|\tilde{g}_k\|\epsilon^{-1/2}, \epsilon^{1/2}\right\}. \end{aligned} \tag{21}$$

Considering now the case $\alpha_k < 1$ and using [44, equation 32, p. 1459], we have:

$$\alpha_k \geq \theta\frac{6}{L_H+\eta}\epsilon_H\|d_k\|^{-1} = \frac{6\theta}{L_H+\eta}\epsilon^{1/2}\|d_k\|^{-1},$$

which leads to

$$\alpha_k\|d_k\| \geq \frac{6\theta}{L_H+\eta}\epsilon^{1/2}. \tag{22}$$

Putting (21) and (22) together, we obtain

$$\begin{aligned}\alpha_k\|d_k\| &\geq \min\left\{\frac{1}{(1+\sqrt{1+L_H/2})^3}, \left[\frac{6\theta}{L_H+\eta}\right]^3\right\}\min\{\|\tilde{g}_k\|\epsilon^{-1/2}, \epsilon^{1/2}\} \\ &= c_{rn}\min\{\|\tilde{g}_k\|\epsilon^{-1/2}, \epsilon^{1/2}\} \geq c\epsilon^{1/2}. \end{aligned}$$

By putting the three cases together, we arrive at the desired conclusion. ■

In order to tackle the worst-case behavior of our method, we provide the following upper bound on the norm of the search directions computed by our method.

Lemma 3.4 *Let Assumption 3.2 hold for a realization of Algorithm 1. Then, for any index k, it holds that*

$$\|d_k\| \leq \max\{U_H, U_g^{1/2}\}. \tag{23}$$

Proof. The bound (23) trivially holds if $\|d_k\| = 0$, thus we only need to prove that it holds for $\|d_k\| > 0$. We consider three disjoint cases:

Case 1: $\lambda_k < -\epsilon^{1/2}$. Then the negative curvature step is taken and $\|d_k\| = |\lambda_k| \leq U_H$.

Case 2: $\lambda_k > \|g_k\|^{1/2}$. We can suppose that $\|g_k\| > 0$ because otherwise $\|d_k\| = 0$. Then, d_k is a Newton step with

$$\|d_k\| \leq \|H_k^{-1}\|\|g_k\| \leq \|g_k\|^{-1/2}\|g_k\| \leq \|g_k\|^{1/2} \leq U_g^{1/2}.$$

Case 3: $-\epsilon^{1/2} \leq \lambda_k \leq \|g_k\|^{1/2}$. As in Case 2, we suppose that $\|g_k\| > 0$ as $\|d_k\| = 0$ if this does not hold. Then, d_k is a regularized Newton step with

$$\begin{aligned} \|d_k\| &= \|(H_k + (\|g_k\|^{1/2} + \epsilon^{1/2})\mathbb{I}_n)^{-1} g_k\| \\ &\leq \|(H_k + (\|g_k\|^{1/2} + \epsilon^{1/2})\mathbb{I}_n)^{-1}\|\|g_k\| \\ &\leq (\lambda_k + \|g_k\|^{1/2} + \epsilon^{1/2})^{-1}\|g_k\|/ \\ &\leq \|g_k\|^{-1/2}\|g_k\| \leq U_g^{1/2}, \end{aligned}$$

where the last line uses the fact that $\lambda_k + \epsilon^{1/2} \geq 0$ and $\|g_k\| > 0$. ■

For future use in this section, we define the following function on $[0, \infty) \times [0, 1]$:

$$\varrho(t, q) \;:=\; \frac{(1-q)U_L}{(1-q)U_L + q\frac{\eta t^3}{24}}, \quad \text{where } U_L := U_g \max\{U_H, U_g^{1/2}\} + \tfrac{L}{2}\max\{U_h^2, U_g\}. \tag{24}$$

We observe that the function ϱ is well-defined, with values in $[0, 1]$, and decreasing in its first and second arguments.

Based on the result of Lemmas 3.3 and 3.4, we can provide a generic guarantee on the expected decrease at every iteration: this is the purpose of Theorem 3.1.

Theorem 3.1 *Let Assumptions 3.1 and 3.2 hold. Suppose also that Assumption 3.4 holds with $\delta = (\delta_f, \delta_g, \delta_H)$ satisfying*

$$\delta_f \leq \frac{\eta}{24} c^3 \epsilon^{3/2}, \quad \delta_g \leq \kappa_g \epsilon, \quad \delta_H \leq \kappa_H \epsilon^{1/2} \tag{25}$$

where $\epsilon > 0$, $\kappa_g \in (0, 1)$, $\kappa_H \in (0, 1)$ and c is chosen as in Lemma 3.3. Finally, consider the following random event

$$E_k^{\text{sta}} = \left\{x_k \text{ is not } ((1+\kappa_g)\epsilon, (1+\kappa_H)\epsilon^{1/2})\text{-function stationary}\right\}. \tag{26}$$

If the sample fraction π_k is chosen such that

$$\pi_k \;\geq\; \varrho(c\epsilon^{1/2}, p), \tag{27}$$

where ϱ is given by (24), then

$$\mathbb{E}\left[f(x_k + \alpha_k d_k) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \;\leq\; -\varrho(c\epsilon^{1/2}, p)\frac{\eta}{24} c^3 \epsilon^{3/2}. \tag{28}$$

Proof. By definition, one has that:

$$\begin{aligned}
&\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \\
&= \mathbb{P}(I_k|\mathcal{F}_{k-1}, E_k^{\text{sta}})\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}, I_k\right] + \\
&\left(1 - \mathbb{P}(I_k|\mathcal{F}_{k-1}, E_k^{\text{sta}})\right)\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}, \overline{I_k}\right] \\
&= \tilde{p}_k\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}, I_k\right] + (1 - \tilde{p}_k)\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}, \overline{I_k}\right] \quad (29)
\end{aligned}$$

in which I_k is the event corresponding to the model being δ-accurate and $\tilde{p}_k := \mathbb{P}(I_k|\mathcal{F}_{k-1}, E_k^{\text{sta}})$ (note that $\tilde{p}_k \geq p$). Since $\mathcal{F}_{k-1} \cap E_k^{\text{sta}} \subset \mathcal{F}_{k-1}$, we can apply the result from [21, Theorem 5.1.6], which states that for any random variable X (not necessarily belonging to $\mathcal{F}_{k-1} \cap E_k^{\text{sta}}$ or $\mathcal{F}_{k-1}$), we have:

$$\mathbb{E}\left[X|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] = \mathbb{E}\left[\mathbb{E}\left[X|\mathcal{F}_{k-1}\right]|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \quad (30)$$

Thus, denoting by $\mathbf{1}(I_k)$ the indicator variable of the random event I_k, we have

$$\begin{aligned}
\tilde{p}_k = \mathbb{P}\left(I_k|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right) &= \mathbb{E}\left[\mathbf{1}(I_k)|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\mathbf{1}(I_k)|\mathcal{F}_{k-1}\right]|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \\
&= \mathbb{E}\left[\mathbb{P}(I_k|\mathcal{F}_{k-1})|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \\
&= \mathbb{E}\left[p_k|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right] = p_k \geq p
\end{aligned}$$

We now first bound the term corresponding to the occurrence of I_k in (29). Under occurrence of I_k and E_k^{sta}, one has:

$$\begin{aligned}
&f(x_k + \alpha_k d_k) - f(x_k) \\
&= f(x_k + \alpha_k d_k) - m_k(x_k + \alpha_k d_k) + m_k(x_k + \alpha_k d_k) - m_k(x_k) + m_k(x_k) - f(x_k) \\
&\leq 2\delta_f + m_k(x_k + \alpha_k d_k) - m_k(x_k) \\
&\leq 2\delta_f - \frac{\eta}{6}\alpha_k^3\|d_k\|^3 \\
&\leq \frac{\eta c^3}{12}\epsilon^{3/2} - \frac{\eta}{6}\alpha_k^3\|d_k\|^3 \;\leq\; -\frac{\eta}{12}\alpha_k^3\|d_k\|^3 \leq -\frac{\eta c^3}{12}\epsilon^{3/2} \;\leq\; -\varrho(c\epsilon^{1/2}, p)\frac{\eta c^3}{12}\epsilon^{3/2}, \quad (31)
\end{aligned}$$

where we used the bound (25) and the fact that $\alpha_k\|d_k\| \geq c\,\epsilon^{1/2}$, which follows from Lemma 3.3 and E_k^{sta}.

We now turn to the second case in (29), for which we exploit the following decomposition:

$$f(x_k + \alpha_k d_k) - f(x_k) = \pi_k\left(m_k(x_k + \alpha_k d_k) - m_k(x_k)\right) + (1 - \pi_k)\left(f_{\mathcal{S}_k^c}(x_k + \alpha_k d_k) - f_{\mathcal{S}_k^c}(x_k)\right),$$

with $f_{\mathcal{S}_k^c} = \frac{1}{N - |\mathcal{S}_k|}\sum_{i \notin \mathcal{S}_k} f_i$. Using the decrease condition (7) to bound the first term, and a first-order Taylor expansion to bound the second term, we obtain:

$$\begin{aligned}
&f(x_k + \alpha_k d_k) - f(x_k) \\
&\leq -\pi_k\frac{\eta}{6}\alpha_k^3\|d_k\|^3 + (1 - \pi_k)\frac{1}{N - |\mathcal{S}_k|}\sum_{i \notin \mathcal{S}_k}\left\{\alpha_k\nabla f_i(x_k)^\top d_k + \frac{L_i}{2}\alpha_k^2\|d_k\|^2\right\} \\
&\leq -\pi_k\frac{\eta}{6}\alpha_k^3\|d_k\|^3 + (1 - \pi_k)\max_{i \notin \mathcal{S}_k}\left\{\alpha_k\nabla f_i(x_k)^\top d_k + \frac{L_i}{2}\alpha_k^2\|d_k\|^2\right\} \\
&\leq -\pi_k\frac{\eta}{6}\alpha_k^3\|d_k\|^3 + (1 - \pi_k)\left\{\alpha_k\|\nabla f_{i_k}(x_k)\|\|d_k\| + \frac{L_{i_k}}{2}\alpha_k^2\|d_k\|^2\right\},
\end{aligned}$$

where $i_k \in \arg\max_{i \notin \mathcal{S}_k} \left\{ \alpha_k \nabla f_i(x_k)^T d_k + \frac{L_i}{2} \alpha_k^2 \|d_k\|^2 \right\}$ and we used the Lipschitz continuity assumption on the functions f_i's. Introducing the constants U_g and L to remove the dependencies on i_k, we further obtain:

$$f(x_k + \alpha_k d_k) - f(x_k) \leq -\pi_k \frac{\eta}{6} \alpha_k^3 \|d_k\|^3 + (1 - \pi_k) \left\{ U_g \alpha_k \|d_k\| + \frac{L}{2} \alpha_k^2 \|d_k\|^2 \right\}.$$

Using now Lemma 3.4, we arrive at:

$$\begin{aligned} f(x_k + \alpha_k d_k) - f(x_k) &\leq (1 - \pi_k) \left\{ U_g \max\{U_H, U_g^{1/2}\} + \frac{L}{2} \max\{U_h^2, U_g\} \right\} \\ &= (1 - \pi_k) U_L \leq \Big(1 - \varrho(c\epsilon^{1/2}, p)\Big) U_L, \end{aligned}$$

where U_L is defined as in (24), and we use (27) to obtain the last inequality.
Putting the two cases together then yields:

$$\mathbb{E}\left[f(x_k + \alpha_k d_k) - f(x_k) | \mathcal{F}_{k-1}, E_k^{\text{sta}} \right] \leq -\tilde{p}_k \, \varrho(c\epsilon^{1/2}, p) \frac{\eta c^3}{12} \epsilon^{3/2} + (1 - \tilde{p}_k) \Big(1 - \varrho(c\epsilon^{1/2}, p)\Big) U_L.$$

Hence,

$$\mathbb{E}\left[f(x_k + \alpha_k d_k) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}} \right] \leq -\varrho(c\epsilon^{1/2}, p) \tfrac{\eta c^3}{24} \epsilon^{3/2}. \tag{32}$$

provided that

$$\varrho(c\epsilon^{1/2}, p) = \frac{(1-p)U_L}{(1-p)U_L + p\frac{\eta c^3}{24}\epsilon^{3/2}} \geq \frac{(1-\tilde{p}_k)U_L}{(1-\tilde{p}_k)U_L + \tilde{p}_k \frac{\eta c^3}{24}\epsilon^{3/2}} = \varrho(c\epsilon^{1/2}, \tilde{p}_k),$$

and this is true because the expression on the right-hand side is a decreasing function in its second argument. The desired result follows. ■

In the next section, we will see how the result of Theorem 3.1 can be used to derive a global convergence rate for Algorithm 1.

4 Global convergence rate and complexity analysis

In this section, we provide a global rate of convergence towards an approximate stationary point in expectation for our method. More precisely, we seek an $((1+\kappa_g)\epsilon, (1+\kappa_H)\epsilon^{1/2})$-function stationary point in the sense of Definition 3.3, that is, an iterate satisfying:

$$\min\{\|\nabla f(x_k)\|, \|\nabla f(x_{k+1})\|\} \leq (1+\kappa_g)\epsilon \quad \text{and} \quad \lambda_{\min}(\nabla^2 f(x_k)) \geq -(1+\kappa_H)\epsilon^{1/2}. \tag{33}$$

Since our method only operates with a (subsampling) model of the objective, we are only able to check whether the current iterate is an $(\epsilon, \epsilon^{1/2})$-*model* stationary point according to Definition 3.3, i.e. an iterate x_k such that

$$\min\{\|g_k\|, \|g_k^+\|\} \leq \epsilon \quad \text{and} \quad \lambda_k \geq -\epsilon^{1/2}.$$

Compared to the general setting of Definition 3.3, we are using $\epsilon_g = \epsilon_H^2 = \epsilon$. This specific choice of first- and second-order tolerances has been observed to yield optimal complexity bounds for

a number of algorithms, in the sense that the dependence on ϵ is minimal (see e.g. [13, 44]). The rules defining what kind of direction (negative curvature, Newton, etc) is chosen at every iteration of Algorithm 1 implicitly rely on this choice.

Our goal is to relate the model stationarity property (4) to its function stationarity counterpart (33). For this purpose, we first establish a general result regarding the expected number of iterations required to reach function stationarity. We then introduce a stopping criterion involving multiple consecutive iterations of model stationarity: with this criterion, our algorithm can be guaranteed to terminate at a function stationary point with high probability. Moreover, the expected number of iterations until this termination occurs is of the same order of magnitude as the expected number of iterations required to reach a function stationary point.

4.1 Expected iteration complexity

The proof of our expected complexity bound relies upon two arguments from martingales and stopping time theory. The first one is a martingale convergence result [40, Theorem 1], which we adapt to our setting in Theorem 4.1.

Theorem 4.1 *Let $(\Omega, \Sigma, \mathbb{P})$ be a probability space. Let $\{\Sigma^k\}_k$ be a sequence of sub-sigma algebras of Σ such that $\Sigma^k \subset \Sigma^{k+1}$.*

If α^k is a positively valued sequence of random variables on Σ, and there is a deterministic sequence $\nu^k \geq 0$ such that,

$$\mathbb{E}[\alpha^{k+1}|\Sigma^k] + \nu^k \leq \alpha^k$$

then α^k converges to a $[0, \infty)$-valued random variable almost surely and $\sum_k \nu^k < \infty$.

At each iteration, Theorem 3.1 guarantees a certain expected decrease for the objective function. Theorem 4.1 will be used to show that such a decrease cannot hold indefinitely if the objective is bounded from below.

The second argument comes from stopping time analysis (see, e.g., [43, Theorem 6.4.1]) and is given in Theorem 4.2. The notations have been adapted to our setting.

Theorem 4.2 *Let T be a stopping time for the process $\{Z_k, k \geq 0\}$ and let $\bar{Z}_k = Z_k$ for $k \leq T$ and $\bar{Z}_k = Z_T$ for $k > T$. If either one of the three properties below hold:*

- *$\bar{Z}_k$ is uniformly bounded;*
- *T is bounded;*
- *$\mathbb{E}[T] < \infty$ and there is an $R < \infty$ such that $\mathbb{E}[|Z_{k+1} - Z_k| \,|\, Z_0, ..., Z_k] < R$;*

Then, $\mathbb{E}\left[\bar{Z}_k\right] \to \mathbb{E}\left[Z_T\right]$. Moreover, $\mathbb{E}[Z_T] \geq \mathbb{E}[Z_0]$ (resp. $\mathbb{E}[Z_T] \leq \mathbb{E}[Z_0]$) if $\{Z_k\}$ is a submartingale (resp. a supermartingale).

Theorem 4.2 enables us to exploit the martingale-like property of Definition 3.2 in order to characterize the index of the first stationary point encountered by the method.

Using both theorems along with Theorem 3.1, we bound the expected number of iterations needed by Algorithm 1 to produce an approximate function stationary point for the model.

Theorem 4.3 *Let Assumptions 3.1, 3.2 and 3.4 hold, with $\delta = (\delta_f, \delta_g, \delta_H)$ satisfying (25). Suppose that for every index k, the sample size π_k satisfies (27).*

Define T_ϵ as the first iteration index k of Algorithm 1 for which

$$\min\{\|\nabla f(x_k)\|, \|\nabla f(x_{k+1})\|\} \leq (1+\kappa_g)\epsilon \quad \text{and} \quad \lambda_{\min}\left(\nabla^2 f(x_k)\right) \geq -(1+\kappa_H)\epsilon^{1/2}.$$

Then, $T_\epsilon < \infty$ almost surely, and

$$\mathbb{E}[T_\epsilon] \leq \begin{cases} \frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\epsilon^{-3/2} + 1 & \text{if } \pi_k = 1 \ \forall k; \\ \frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\, \varrho(c\epsilon^{1/2}, p)^{-1}\, \epsilon^{-3/2} + 1 & \text{otherwise,} \end{cases} \tag{34}$$

where $\hat{c} = \frac{\eta}{24}c^3$.

Proof. Since both x_{k+1} and $\mathcal{S}_k$ belong to $\mathcal{F}_k$, we have $\{T_\epsilon = k\} \in \mathcal{F}_k$ for all k, thus T_ϵ is indeed a stopping time.

We first show that the event $\{T_\epsilon = \infty\}$ has a zero probability of occurrence. To this end, we suppose that for every iteration index k, we have $k < T_\epsilon$. In particular, this means that the events $E_0^{\text{sta}}, \ldots, E_k^{\text{sta}}$ occur, where we recall that $E_j^{\text{sta}} \in \mathcal{F}_{j-1}$. We thus define the following filtration:

$$\mathcal{T}_0 = \mathcal{F}_{-1} \cap E_0^{\text{sta}}, \quad \mathcal{T}_k = \mathcal{F}_{k-1} \cap \left(E_0^{\text{sta}} \cap \cdots \cap E_k^{\text{sta}}\right) \ \forall k \geq 1, \tag{35}$$

where we use $\mathcal{F} \cap E$ to denote the trace σ-algebra of the event E on the σ-algebra $\mathcal{F}$, i.e., $\mathcal{F} \cap E = \{E \cap F \ : \ F \in \mathcal{F}\}$. For every $k \geq 0$ and any event A, we thus have by the same argument used to establish (30) that

$$\mathbb{E}\left[A|\mathcal{T}_k\right] = \mathbb{E}\left[\mathbb{E}\left[A|\mathcal{F}_{k-1}, E_k^{\text{sta}}\right]|\mathcal{T}_k\right]$$

and we also have $\mathcal{T}_k \subset \mathcal{T}_{k+1}$.

For notational convenience, we define

$$c_\epsilon = \begin{cases} \hat{c}\epsilon^{3/2} & \text{if } \pi_k = 1 \ \forall k; \\ \hat{c}\, \varrho(c\epsilon^{1/2}, p)\epsilon^{3/2} & \text{otherwise.} \end{cases} \tag{36}$$

If $T_\epsilon = \infty$, then the assumptions of Theorem 3.1 are satisfied for all iterations k. In particular, for all k, event E_k^{sta} occurs and $\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \leq -c_\epsilon$. Recalling the definition of $\mathcal{T}_k$ (35), we get

$$\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{T}_k\right] = \mathbb{E}\left[\mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{F}_{k-1}, E_k^{\text{sta}}\right] \mid \mathcal{T}_k\right] \leq \mathbb{E}\left[-c_\epsilon \mid \mathcal{T}_k\right] = -c_\epsilon.$$

Thus,

$$\begin{aligned} \mathbb{E}\left[f(x_{k+1}) - f(x_k) \mid \mathcal{T}_k\right] &\leq -c_\epsilon \\ \mathbb{E}\left[f(x_{k+1}) \mid \mathcal{T}_k\right] + c_\epsilon &\leq \mathbb{E}\left[f(x_k) \mid \mathcal{T}_k\right] \\ \mathbb{E}\left[f(x_{k+1}) \mid \mathcal{T}_k\right] + c_\epsilon &\leq f(x_k), \end{aligned}$$

where the last relation comes from the fact that $x_k \in \mathcal{T}_k$. Subtracting f_{low} on both sides yields:

$$\mathbb{E}\left[f(x_{k+1}) - f_{\text{low}} \mid \mathcal{T}_k\right] + c_\epsilon \leq f(x_k) - f_{\text{low}}. \tag{37}$$

As a result, we can apply Theorem 4.1 with $\alpha^k = f(x_k) - f_{\text{low}} \geq 0$, $\Sigma^k = \mathcal{T}_k$ and $\nu^k = c_\epsilon > 0$: we thus obtain that $\sum_{k=0}^{\infty} c_\epsilon < \infty$, which is obviously false. This implies that T_ϵ must be finite almost surely.

Consider now the sequence of random variables given by,

$$R_k = f(x_{\min(k,T_\epsilon)}) + \max\left(\min(k, T_\epsilon) - 1, 0\right) c_\epsilon.$$

Theorem 3.1 ensures that for any $k < T_\epsilon$, one has

$$\begin{aligned} \mathbb{E}[R_{k+1} \mid \mathcal{T}_k] &= \mathbb{E}[f(x_{k+1}) \mid \mathcal{T}_k] + k\, c_\epsilon \\ &\leq f(x_k) - c_\epsilon + k\, c_\epsilon \leq R_k \end{aligned}$$

and for $k \geq T_\epsilon$, trivially $R_{k+1} = R_k$, and thus R_k is a supermartingale. Moreover, since in $\mathcal{T}_k$, it holds that $T_\epsilon \geq k + 1$,

$$\begin{aligned} \mathbb{E}\left[|R_{k+1} - R_k| \mid \mathcal{T}_k\right] &= \mathbb{E}\left[|f(x_{k+1}) + (k+1)c_\epsilon - f(x_k) - kc_\epsilon| \mid \mathcal{T}_k\right] \\ &\leq \mathbb{E}\left[|f(x_{k+1}) - f(x_k)| + c_\epsilon \mid \mathcal{T}_k\right] \\ &\leq c_\epsilon + \max\left(c_\epsilon, f_{\max} - f_{\text{low}}\right) < \infty \end{aligned}$$

thus the expected increment between R_k and R_{k+1} is bounded. Since $T_\epsilon < \infty$ almost surely, we satisfy the assumptions of Theorem 4.2: it thus holds that $\mathbb{E}[R_{T_\epsilon}] \leq \mathbb{E}[R_0]$ which implies that,

$$f_{\text{low}} + (\mathbb{E}\left[T_\epsilon\right] - 1)c_\epsilon \leq \mathbb{E}\left[R_{T_\epsilon}\right] \leq \mathbb{E}[R_0] = f(x_0)$$

and thus,

$$\mathbb{E}\left[T_\epsilon\right] \leq \frac{(f(x_0) - f_{\text{low}})}{c_\epsilon} + 1.$$

Replacing c_ϵ by its definition gives the desired bound. ■

The result of Theorem 4.3 gives a worst-case complexity bound on the expected number of iterations until a function stationarity point is reached. This does not provide a practical stopping criterion for the algorithm, because only *model* stationarity can be tested for during an algorithmic run (even though in the case of $\pi_k = p = 1$, both notions of stationarity are equivalent). Still, by combining Theorem 4.3 and Lemma 3.1, we can show that after at most $\mathcal{O}(\varrho(c\epsilon^{1/2}, p)\epsilon^{-3/2})$ iterations on average, if the model is accurate, then the corresponding iterate will be function stationary.

In our algorithm, we assume that accuracy is only guaranteed with a certain probability at every iteration. As a result, stopping after encountering an iterate that is model stationary only comes with a weak guarantee of returning a point that is function stationary. In developing an appropriate stopping criterion, we wish to avoid such "false positives". To this end, one possibility consists of requiring model stationarity to be satisfied for a certain number of successive iterations. Our approach is motivated by the following result.

Proposition 4.1 *Under the assumptions of Theorem 4.3, suppose that Algorithm 1 reaches an iteration index $k + J$ such that for every $j \in \{k, k+1, \ldots, k+J\}$,*

$$\min\left\{\|g_j\|, \|g_j^+\|\right\} \leq \epsilon \quad \text{and} \quad \lambda_j \geq -\epsilon^{1/2}.$$

Suppose further that δ_g and δ_H satisfy (14), and that the sample sizes are selected independently of the current iterate. Then, with probability at least $1-(1-p)^{J+1}$, where p is the lower bound on p_k given by Assumption 3.4, one of the iterates $\{x_k, x_{k+1}, \dots, x_{k+J}\}$ is $((1+\kappa_g)\epsilon, (1+\kappa_H)\epsilon^{1/2})$-function stationary.

Proof. In this proof, we will use the notation $\mathbb{P}_k(\dots) = \mathbb{P}(\cdot|\mathcal{F}_{k-1})$. Recall the event I_j from Definition 3.2. We also consider the random events

$$E \;=\; \left\{\text{One of the iterates in } \{x_{k+j}\}_{j=0..J} \text{ is } ((1+\kappa_g)\epsilon, (1+\kappa_H)\epsilon^{1/2})\text{-function stationary}\right\},$$

and

$$\forall j = 0, \dots, J, \quad E_j \;=\; \left\{\text{The iterate } x_{k+j} \text{ is } (\epsilon, \epsilon^{1/2})\text{-model stationary}\right\}.$$

For every $j = 0, \dots, J$, we have $E_j \in \mathcal{F}_{k+j}$ and $I_{k+j} \in \mathcal{F}_{k+j}$. Moreover, the event E_j and I_{k+j} are conditionally independent, i.e. for every $j = 0, \dots, J$,

$$\mathbb{P}_k\left(E_j \cap I_{k+j} | \mathcal{F}_{k+j-1}\right) \;=\; \mathbb{P}_k\left(E_j | \mathcal{F}_{k+j-1}\right) \mathbb{P}_k\left(I_{k+j} | \mathcal{F}_{k+j-1}\right). \tag{38}$$

This conditional independence holds because $x_{k+j} \in \mathcal{F}_{k+j-1}$, and the model m_{k+j} is constructed independently of x_{k+j} by assumption.

The statement of the theorem corresponds to

$$\mathbb{P}_k\left(E | E_0, \dots, E_J\right) \;\geq\; 1 - (1-p)^{J+1},$$

which is what we want to prove. By Lemma 3.1,

$$\mathbb{P}_k\left(E | E_0, \dots, E_J\right) \;\geq\; \mathbb{P}_k\left(\bigcup_{0\leq j\leq J} I_{k+j} \,\middle|\, E_0, \dots, E_J\right) = 1 - \mathbb{P}_k\left(\bigcap_{0\leq j\leq J} \bar{I}_{k+j} \,\middle|\, E_0, \dots, E_J\right).$$

and it thus suffices to prove that

$$\mathbb{P}_k\left(\bigcap_{0\leq j\leq J} \bar{I}_{k+j} \,\middle|\, E_0, \dots, E_J\right) \;\leq\; (1-p)^{J+1} \tag{39}$$

to obtain the desired result.

We now make use of the probabilistically accuracy property. For every $j = 0, \dots, J$, we have

$$\mathbb{P}_k\left(I_{k+j} | A\right) \geq p, \tag{40}$$

for any set of events A belonging to the σ-algebra $\mathcal{F}_{k+j-1}$ [21, Chapter 5]. In particular, for any $j \geq 1$, $\mathbb{P}_k\left(I_{k+j} | I_k, \dots, I_{k+j-1}, E_0, \dots, E_j\right) \geq p$.

Returning to our target probability, we have:

$$
\begin{aligned}
&\mathbb{P}_k\left(\bigcap_{0\leq j\leq J}\bar{I}_{k+j}\middle|E_0,\ldots,E_J\right)\\
&=\frac{\mathbb{P}_k\left(\left\{\bigcap_{0\leq j\leq J}\bar{I}_{k+j}\right\}\cap E_J\middle|E_0,\ldots,E_{J-1}\right)}{\mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1}\right)}\\
&=\frac{\mathbb{P}_k\left(\bar{I}_J\cap E_J|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)\mathbb{P}_k\left(\cap_{0\leq j\leq J-1}\bar{I}_{k+j}\middle|\ E_0,\ldots,E_{J-1}\right)}{\mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1}\right)}\\
&=\mathbb{P}_k\left(\bar{I}_J\middle|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)\mathbb{P}_k\left(\cap_{0\leq j\leq J-1}\bar{I}_{k+j}\middle|\ E_0,\ldots,E_{J-1}\right)\\
&\times\frac{\mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)}{\mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1}\right)}\\
&\leq\mathbb{P}_k\left(\bar{I}_J\middle|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)\mathbb{P}_k\left(\cap_{0\leq j\leq J-1}\bar{I}_{k+j}\middle|\ E_0,\ldots,E_{J-1}\right).
\end{aligned}
$$

where the last equality comes from (38), and the final inequality uses the fact that the events $E_0,\ldots,E_{J-1}$ and $I_k,\ldots,I_{k+J-1}$ are pairwise independent, thus

$$
\mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)\ =\ \mathbb{P}_k\left(E_J|E_0,\ldots,E_{J-1}\right).
$$

Using (40), we then have that

$$
\mathbb{P}_k\left(\bar{I}_J\middle|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)=1-\mathbb{P}_k\left(I_J|E_0,\ldots,E_{J-1},I_k,\ldots,I_{k+J-1}\right)\leq 1-p.
$$

Thus,

$$
\mathbb{P}_k\left(\bigcap_{0\leq j\leq J}\bar{I}_{k+j}\middle|E_0,\ldots,E_J\right)\leq(1-p)\mathbb{P}_k\left(\bigcap_{0\leq j\leq J-1}\bar{I}_{k+j}\middle|E_0,\ldots,E_{J-1}\right). \tag{41}
$$

By a recursive argument on the right-hand side of (41), we thus arrive at (39), which yields the desired conclusion. ■

The result of Proposition 4.1 is only of interest if we can ensure that such a sequence of model stationary iterates can occur in a bounded number of iterations. This will be the case provided we reject iterates for which we cannot certify sufficient decrease, hence the following additional assumption.

Assumption 4.1 *In Algorithm 1, Step 7 is replaced by:*
7'. If $\min\{\|g_k\|,\|g_k^+\|\}<\epsilon$ and $\lambda_k>-\epsilon^{1/2}$, set $x_{k+1}=x_k$, otherwise set $x_{k+1}=x_k+\alpha_k\,d_k$.
Moreover, the sample size is selected independently of the current iterate.

This algorithmic change allows us to measure stationarity at a given iterate based on a series of samples. Under a slightly stronger set of assumptions, Proposition 4.2 then guarantees that sequences of model stationary points of arbitrary length will occur in expectation.

Proposition 4.2 *Let Assumptions 3.1, 3.2, 3.4 and 4.1 hold, where δ satisfies (25) with $\kappa_g,\kappa_H\in(0,1)$, and $p_k=p\ \forall k$. For a given $J\in\mathbb{N}$, define $T^m_{\epsilon,J}$ as the first iteration index of Algorithm 1 for which*

$$
\min\{\|g_k\|,\|g_k^+\|\}<\epsilon\quad\textit{and}\quad\lambda_k>-\epsilon^{1/2},\quad\forall k\in\{T^m_{\epsilon,J},T^m_{\epsilon,J}+1,\ldots,T^m_{\epsilon,J}+J\}. \tag{42}
$$

Suppose finally that for every index k, the sample size π_k satisfies

$$\forall k, \quad \pi_k \geq \varrho(c\hat{\epsilon}^{1/2}, p), \tag{43}$$

where $\hat{\epsilon} = \min\{\frac{1-\kappa_g}{1+\kappa_g}, \frac{(1-\kappa_H)^2}{(1+\kappa_H)^2}\}\epsilon$. *(Note that* $\varrho(c\hat{\epsilon}^{1/2}, p) \geq \varrho(c\epsilon^{1/2}, p)$.*)*

Then, $T^m_{\epsilon,J} < \infty$ *almost surely, and*

$$\mathbb{E}[T^m_{\epsilon,J}] \leq \begin{cases} \frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\epsilon^{-3/2} + J + 1 & \text{if } \pi_k = 1 \ \forall k; \\ p^{-(J+1)}\left[\frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\,\varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + J + 1\right] & \text{otherwise.} \end{cases} \tag{44}$$

Proof. As in Theorem 4.3, $T^m_{\epsilon,J}$ clearly is a stopping time. Moreover, if $\pi_k = 1$ for all k, then $T^m_{\epsilon,J} = T_\epsilon$ for every J, where T_ϵ is the stopping time defined in Theorem 4.3, and therefore the result holds. In what follows, we thus focus on the remaining case.

Consider an iterate k such that x_k is $(\hat{\epsilon}, \hat{\epsilon}^{1/2})$-function stationary and the model m_k is accurate. From the definition of $\hat{\epsilon}$, such an iterate is also $((1-\kappa_g)\epsilon, (1-\kappa_H)\epsilon^{1/2})$-function stationary and the model m_k is accurate. Then, by a reasoning similar to that of the proof of Lemma 3.1, we can show that x_k is $(\epsilon, \epsilon^{1/2})$-model stationary. As a result, if $T^m_{\epsilon,J} > k$, one of the models $m_k, m_{k+1}, \ldots, m_{k+J}$ must be inaccurate, which happens with probability $1 - p^{J+1}$.

Let $T_{\epsilon,J}$ be the first iteration index for which the iterate is a $(\hat{\epsilon}, \hat{\epsilon}^{1/2})$ function stationary point and satisfies (42). Clearly $T^m_{\epsilon,J} \leq T_{\epsilon,J}$ (for all realizations of these two stopping times), and it thus suffices to bound $T_{\epsilon,J}$ in expectation. By applying Theorem 4.3 (with ϵ in the theorem's statement replaced by $\hat{\epsilon}$), one can see that there must exist an infinite number of $(\hat{\epsilon}, \hat{\epsilon}^{1/2})$-function stationary points in expectation. More precisely, letting $\{T^{(i)}_{\hat{\epsilon}}\}_{i=1,\ldots}$ be the corresponding stopping times indicating the iteration indexes of these points and using Theorem 4.3 (excluding the case $\pi_k = 1\ \forall k$), we have

$$\begin{aligned} \mathbb{E}\left[T^{(1)}_{\hat{\epsilon}}\right] = \mathbb{E}\left[T_{\hat{\epsilon}}\right] &\leq \frac{(f(x_0) - f_{\text{low}})}{\hat{c}}\,\varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + 1, \\ \forall i \geq 1, \quad \mathbb{E}\left[T^{(i+1)}_{\hat{\epsilon}} - T^{(i)}_{\hat{\epsilon}}\right] &\leq \frac{(f(x_0) - f_{\text{low}})}{\hat{c}}\,\varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + 1. \end{aligned}$$

Consider now the subsequence $\{T^{(i_\ell)}_{\hat{\epsilon}}\}_{\ell=1,\ldots}$ such that all stopping times are at least J iterations from each other, i.e., for every $\ell \geq 1$, we have $T^{(i_{\ell+1})}_{\hat{\epsilon}} - T^{(i_\ell)}_{\hat{\epsilon}} \geq J$. For such a sequence, we get

$$\begin{aligned} \forall \ell \geq 1, \quad \mathbb{E}\left[T^{(i_{\ell+1})}_{\hat{\epsilon}} - T^{(i_\ell)}_{\hat{\epsilon}}\right] &\leq \frac{(f(x_0) - f_{\text{low}})}{\hat{c}}\,\varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + J + 1 \triangleq K(\epsilon, J) \\ \mathbb{E}\left[T^{(i_1)}_{\hat{\epsilon}}\right] = \mathbb{E}\left[T_{\hat{\epsilon}}\right] &\leq \frac{(f(x_0) - f_{\text{low}})}{\hat{c}}\,\varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + 1 \leq K(\epsilon, J). \end{aligned}$$

For every $\ell \geq 1$, we define the event

$$B_\ell = \bigcap_{j=0}^{J} I_{T^{(i_\ell)}_{\hat{\epsilon}}+j} = \bigcap_{j=0}^{J} \left\{ m_{T^{(i_\ell)}_{\hat{\epsilon}}+j} \text{ is accurate} \right\}.$$

By assumption, the models are generated independently of the current iterate, and for every k, $\mathbb{P}(I_k|\mathcal{F}_{k-1}) = p$. By the same recursive reasoning as in the proof of Proposition 4.1, we have

that $\mathbb{P}\left(B_\ell | \mathcal{F}_{T_{\hat{\epsilon}}^{(i_\ell)}-1}\right) = p^{J+1}$. Moreover, by definition of the sequence $\{T_{\hat{\epsilon}}^{(i_\ell)}\}$, two stopping times in that sequence correspond to two iteration indexes distant of at least $J+1$. Therefore, they also correspond to two separate sequences of $(J+1)$ models that are generated in an independent fashion. We can thus consider $\{B_\ell\}$ to be an independent sequence of Bernoulli trials. Therefore, the variable G representing the number of runs of B_ℓ until success follows a geometric distribution with an expectation less than $\frac{1}{p^{J+1}} = p^{-(J+1)} < \infty$. On the other hand, $T_{\epsilon,J}$ is less than the first element of $\{T_{\hat{\epsilon}}^{(i_\ell)}\}$ for which B_ℓ happens, and thus $T_{\epsilon,J} \leq T_{\hat{\epsilon}}^{(i_G)}$. To conclude the proof, we define

$$S_G = T_{\hat{\epsilon}}^{(i_G)}, \qquad X_1 = T_{\hat{\epsilon}}^{(i_1)} = T_{\hat{\epsilon}}^{1}, \quad X_\ell = T_{\hat{\epsilon}}^{(i_\ell)} - T_{\hat{\epsilon}}^{(i_{\ell-1})} \quad \forall \ell \geq 2.$$

From the proof of Wald's equation [21, Theorem 4.1.5] (more precisely, from the third equation appearing in that proof), one has

$$\mathbb{E}\left[S_G\right] = \sum_{\ell=1}^{\infty} \mathbb{E}\left[X_\ell\right] \mathbb{P}(G \geq \ell).$$

Since $\mathbb{E}\left[X_\ell\right] \leq K(\epsilon, J)$, one arrives at

$$\mathbb{E}\left[T_{\epsilon,J}^m\right] \leq \mathbb{E}\left[T_{\epsilon,J}\right] \leq \mathbb{E}\left[T_{\hat{\epsilon}}^{(i_G)}\right] \leq K(\epsilon, J) \sum_{\ell=1}^{\infty} \mathbb{P}(G \geq \ell) = K(\epsilon, J)\mathbb{E}\left[G\right],$$

which is the desired result. ■

With the result of Proposition 4.2, we are guaranteed that there will exist consecutive iterations satisfying model stationarity in expectation. Checking for stationarity over successive iterations thus represents a valid stopping criterion in practice. If an estimate of the probability p is known, one can even choose J to guarantee that the probability of computing a stationary iterate is sufficiently high.

Evaluation complexity: As a final note to this section, we observe that the number of evaluations of f_i and their derivatives is bounded in expectation, thanks to the result of Theorem 4.3. Note that we must account for the additional objective evaluations induced by the line-search process (see [44, Theorem 8] for details).

Corollary 4.1 *Under the assumptions of Proposition 4.2, assuming that the sample size condition (27) is satisfied with $\pi_k \in \{\varrho(c\hat{\epsilon}^{1/2}, p), 1\}$ for every k, the expected number of evaluations of ∇f_i and $\nabla^2 f_i$ are respectively bounded above by*

$$\begin{cases} \frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\epsilon^{-3/2} + 1 & \text{if } \pi_k = 1 \ \forall k; \\ p^{-(J+1)}\left[\frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\ \varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + J + 1\right] & \text{otherwise,} \end{cases} \tag{45}$$

while the expected number of function evaluations is bounded above by

$$(1 + \max\{j_{nc}, j_n, j_{rn}\}) \times \begin{cases} \frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\epsilon^{-3/2} + 1 & \text{if } \pi_k = 1 \ \forall k; \\ p^{-(J+1)}\left[\frac{(f(x_0)-f_{\text{low}})}{\hat{c}}\ \varrho(c\hat{\epsilon}^{1/2}, p)^{-1}\,\hat{\epsilon}^{-3/2} + J + 1\right] & \text{otherwise.} \end{cases} \tag{46}$$

The bounds (45) and (46) match their deterministic counterparts (see [44]) in the worst case.

4.2 Illustration of sample and evaluation complexity for uniform sampling

To conclude this theoretical section, we now comment on the two main requirements made in the analysis of the previous section are related to the function value accuracy δ_f and the sample size π_k. More precisely, for a given tolerance ϵ, we required in Theorem 4.3 that:

1. $\delta_f \leq \frac{\eta}{24} c^3 \epsilon^{3/2}$, $\delta_g \leq \kappa_g \epsilon$, $\delta_H \leq \kappa_H \epsilon^{1/2}$.
2. $\pi_k \geq \varrho(c\epsilon^{1/2}, p) = \dfrac{(1-p)\left(U_g \max\{U_H, U_g^{1/2}\} + \frac{L}{2}\max\{U_h^2, U_g\}\right)}{(1-p)\left(U_g \max\{U_H, U_g^{1/2}\} + \frac{L}{2}\max\{U_h^2, U_g\}\right) + p\frac{\eta}{24} c^3 \epsilon^{3/2}}$.

In this section, we provide estimates of the minimum number of samples necessary to achieve those two conditions in the case of a uniform sampling strategy. To facilitate the exposure, we discard the case $p = 1$, for which the properties above trivially hold, and focus on $p \in (0, 1)$. Although we focus on the properties required for Theorem 4.3, note that a similar analysis holds for the requirements of Proposition 4.2.

For the rest of the section, we suppose that the set $\mathcal{S}_k$ is formed of $n\bar{\pi}$ indexes chosen uniformly at random with replacement. where $\bar{\pi}$ is independent of k. That is, for every $i \in \mathcal{S}_k$ and every $j = 1, \ldots, N$, we have $\mathbb{P}(i = j) = \frac{1}{N}$. This case has been well studied in the case of subsampled Hessian [50] and subsampled gradient [42]. The lemmas below summarize the results.

Lemma 4.1 *Under Assumption 3.2, consider an iterate x_k of Algorithm 1. For any $p \in (0, 1)$, if the sample set $\mathcal{S}_k$ is chosen to be of size*

$$\bar{\pi} \geq \frac{1}{N} \frac{16L^2}{\delta_H} \ln\left(\frac{2N}{1-p}\right), \tag{47}$$

then

$$\mathbb{P}\left(\left\|H(x_k; \mathcal{S}_k) - \nabla^2 f(x_k)\right\| \leq \delta_H \middle| \mathcal{F}_{k-1}\right) \geq p.$$

Proof. See [50, Lemma 4]; note that here we are using L_i (Lipschitz constant of ∇f_i) as a bound on $\|\nabla^2 f_i(x_k)\|$, and that we are providing a bound on the sampling fraction $\bar{\pi} = \frac{|\mathcal{S}_k|}{N}$. See also [47, Theorem 1.1], considering the norm as related to the maximum singular vector. ■

By the same reasoning as for Lemma 4.1, but in one dimension, we can readily provide a sample size bound for obtaining accurate function values. To this end, we define

$$f_{\text{up}} \geq \max_k \max_{i=1,..,N} f_i(x_k). \tag{48}$$

Note that such a bound necessarily exists when the iterates are contained in a compact set. Specific structure of the problem can also guarantee such a bound, even tough it may exhibit dependencies on the problem's dimension. For instance, in the case of classification and logistic regression, one has $f_{\text{up}} = 1$, while in the case of (general) regression, one has $f_{\text{up}} \leq C_1 + C_2\|x\|^2 = \mathcal{O}(n)$. We emphasize that both of these bounds can be very pessimistic.

Lemma 4.2 *Under Assumption 3.2, consider an iterate x_k of Algorithm 1. For any $p \in (0,1)$, if the sample set $\mathcal{S}_k$ is chosen to be of size*

$$\bar{\pi} \geq \frac{1}{N} \frac{16 f_{\text{up}}^2}{\delta_f} \ln\left(\tfrac{2}{1-p}\right), \tag{49}$$

then

$$\mathbb{P}\left(|m(x_k;\mathcal{S}_k) - f(x_k)| \leq \delta_f | \mathcal{F}_{k-1}\right) \geq p.$$

Proof. The proof follows that of [50, Lemma 4.1] by considering $m(x_k;\mathcal{S}_k)$ and $f(x_k)$ as one-dimensional matrices. ■

Lemma 4.3 *Under Assumption 3.2, consider an iterate x_k of Algorithm 1. For any $p \in (0,1)$, if the sample set $\mathcal{S}_k$ is chosen to be of size*

$$\bar{\pi} \geq \frac{1}{N} \frac{U_g^2}{\delta_g^2} \left[1 + \sqrt{8 \ln\left(\tfrac{1}{1-p}\right)}\right]^2, \tag{50}$$

then

$$\mathbb{P}\left(\|g(x_k;\mathcal{S}_k) - \nabla f(x_k)\| \leq \delta_g | \mathcal{F}_{k-1}\right) \geq p.$$

Proof. See [41, Lemma 2]. ■

We now combine the three lemmas to obtain an overall result indicating the required $\bar{\pi}$ such that Lemmas 4.1, 4.2 and 4.3 simultaneously hold, i.e., the event I_k holds.

Theorem 4.4 *Let Assumptions 3.1 and 3.2 hold. For any $p \in (0,1)$, let $\hat{p} = \frac{p+3}{4}$. Suppose that the sample fractions π_k of Algorithm 1 are chosen to satisfy $\pi_k \geq \pi(\epsilon)$ for every k, where*

$$\pi(\epsilon) := \frac{1}{N} \max\left\{ N\ \varrho(c\epsilon^{1/2}, \hat{p}), \frac{344 f_{\text{up}}^2}{\eta c^3 \epsilon^{3/2}} \ln\left(\tfrac{2}{1-\hat{p}}\right), \frac{U_g^2}{\kappa_g^2 \epsilon^2}\left[1 + \sqrt{8\ln\left(\tfrac{1}{1-\hat{p}}\right)}\right]^2, \frac{16 L^2}{\kappa_H \epsilon^{1/2}} \ln\left(\tfrac{2N}{1-\hat{p}}\right) \right\}.$$

Then, the model sequence is p-probabilistically $(\delta_f, \delta_g, \delta_H)$-accurate with $\delta_f = \frac{\eta}{24} c^3 \epsilon^{3/2}$, $\delta_g = \kappa_g \epsilon$ and $\delta_H = \kappa_H \epsilon^{1/2}$. Moreover, all the results from Section 4.1 hold.

Proof. Indeed, let $\delta_f = \frac{\eta}{24} c^3 \epsilon^{3/2}$, $\delta_g = \kappa_g \epsilon$, $\delta_H = \kappa_H \epsilon^{1/2}$ and note that,

$$I_k \equiv I_k^h \cap I_k^g \cap I_k^f,$$

where

$$\begin{aligned} I_k^h &:= \{\left\|H(x_k;\mathcal{S}_k) - \nabla^2 f(x_k)\right\| \leq \delta_H\} \\ I_k^g &:= \{\|g(x_k;\mathcal{S}_k) - \nabla f(x_k)\| \leq \delta_g\} \\ I_k^f &:= \{|m(x_k;\mathcal{S}_k) - f(x_k)| \leq \delta_f\}. \end{aligned}$$

Using the required conditions on π_n, Lemmas 4.1, 4.2 and 4.3 imply

$$\mathbb{P}\left((I_k^f)^c \middle| \mathcal{F}_{k-1}\right) \leq 1 - \hat{p}\ , \quad \mathbb{P}\left((I_k^g)^c \middle| \mathcal{F}_{k-1}\right) \leq 1 - \hat{p}, \quad \text{and} \quad \mathbb{P}\left((I_k^h)^c \middle| \mathcal{F}_{k-1}\right) \leq 1 - \hat{p}.$$

In the other hand, one has

$$\begin{aligned}
\mathbb{P}\left((I_k)^c|\mathcal{F}_{k-1}\right) &= \mathbb{P}\left((I_k^f)^c\middle|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^g)^c|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^g)^c\middle|\mathcal{F}_{k-1}\right) - \mathbb{P}\left((I_k^f)^c \cap (I_k^g)^c\middle|\mathcal{F}_{k-1}\right) \\
&\quad -\mathbb{P}\left((I_k^g)^c \cap (I_k^h)^c|\mathcal{F}_{k-1}\right) - \mathbb{P}\left((I_k^f)^c \cap (I_k^h)^c\middle|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^f)^c \cap (I_k^g)^c \cap (I_k^h)^c\middle|\mathcal{F}_{k-1}\right) \\
&\leq \mathbb{P}\left((I_k^f)^c\middle|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^g)^c|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^h)^c|\mathcal{F}_{k-1}\right) \\
&\quad +\mathbb{P}\left((I_k^f)^c\middle|(I_k^h)^c,(I_k^g)^c,\mathcal{F}_{k-1}\right)\mathbb{P}\left((I_k^h)^c|(I_k^g)^c,\mathcal{F}_{k-1}\right)\mathbb{P}\left((I_k^g)^c|\mathcal{F}_{k-1}\right) \\
&\leq \mathbb{P}\left((I_k^f)^c\middle|\mathcal{F}_{k-1}\right) + 2\mathbb{P}\left((I_k^g)^c|\mathcal{F}_{k-1}\right) + \mathbb{P}\left((I_k^h)^c|\mathcal{F}_{k-1}\right) \leq 4(1-\hat{p}).
\end{aligned}$$

Hence,

$$\mathbb{P}(I_k|\mathcal{F}_{k-1}) \geq 1 - 4(1-\hat{p}) = p,$$

meaning that the model sequence is p-probabilistically $(\delta_f, \delta_g, \delta_H)$-accurate, thus results from Section 4.1 hold. ■

We observe that explicit computation of these bounds would require estimating L, L_H, U_H, and U_g. If the orders of magnitude for the aforementioned quantities are available, they can be used for choosing the sample size.

Comparison with the deterministic line-search method In general, we can consider that at each iteration we must perform $|\mathcal{S}_k|n$ computations to evaluate the sample gradient, $|\mathcal{S}_k|n^2$ to compute the sample Hessian, and evaluate a function $\bar{j}|\mathcal{S}_k|$ times during the line search. Per Lemma 3.3, the maximum number of line search iterations $\bar{j}$ can be considered of the order $\log(1/\epsilon)$. As a result, each iteration requires $|\mathcal{S}_k|(n + n^2 + \log(1/\epsilon))$ computations. For our sampling rate to yield improvement over the deterministic (fully sampled) method, we require $N\log(1/\epsilon)$ to be much larger than $|\mathcal{S}_k|n^2$, implying that the appropriate regime for this algorithm is one where the number of variables is considerably less than the number of data points.

The cost of our method could be improved in several ways. First, by employing inexact linear algebra routines in computing the negative curvature and Newton directions, the cost of Hessian-related computations (only requiring Hessian-vector products) would only be now proportional to n instead of n^2. Secondly, a more practical approach for incorporating subsampling information would consist in approximating second-order information using a smaller sampling size than what is used for approximating first-order information [12]. Both these techniques have a practical motivation, and we believe that they can also be supported by theory. However, our focus in this work is the study of algorithms for which at every iteration, one sample is used to estimate the objective and the derivatives.

Comparison with other sample complexities By comparing it with sample complexity results available in the literature, we can position our method within the existing landscape of results, and get insight about the cost of second-order requirements, as well as that of using inexact function values.

When applied to nonconvex problems, a standard stochastic gradient approach with fixed step size has a complexity in $\mathcal{O}(\epsilon^{-4})$ (both in terms of iterations and gradient evaluations) for reaching approximate first-order optimality [23]. Modified SGD methods that take curvature information can significantly improve over that bound, and even possess second-order guarantees.

Typical guarantees are provided in terms of function stationarity (though this condition cannot be checked in practice), and hold with high probability. Our results hold in expectation, but involve a model stationary condition that we can check at every iteration. Moreover, it is possible to convert a rate in expectation into high-probability rates (of same order), following for instance an argument used for trust-region algorithms with probabilistic models [24].

It is also interesting to compare our complexity orders with those obtained by first-order methods in stochastic optimization. Stochastic trust-region methods [17, 31] require $\mathcal{O}\left(\Delta_k^{-4}\right)$ samples per iteration, where Δ_k is the trust-region radius and serves as an approximation of the norm of the gradient. Similarly, the line-search algorithm of Paquette and Scheinberg [38] guarantees sufficient accuracy in the function values if the sample size is of order $\mathcal{O}\left(\alpha_k^{-2}\|g_k\|^{-4}\right)$ (we use our notations for consistency), which for our method roughly corresponds to $\mathcal{O}\left((\alpha_k\|d_k\|)^{-2}\right)$. Our sample complexity is of order $\mathcal{O}\left((\alpha_k\|d_k\|)^{-3}\right)$ but for second-order convergence rates. One could thus conjecture that endowing the algorithm of [38] with second-order guarantees in expectation would increase the sample complexity from $\mathcal{O}\left((\alpha_k\|d_k\|)^{-2}\right)$ to $\mathcal{O}\left((\alpha_k\|d_k\|)^{-3}\right)$.

Finally, we discuss sample bounds for Newton-based methods in a subsampling context. In [50], it was shown that the desired complexity rate of $\mathcal{O}(\epsilon^{-3/2})$ is achieved with probability $1-p$, if for every k, the sample size (for approximating the Hessian) satisfies:

$$|\mathcal{S}_k| \geq \frac{16U_H^2}{\epsilon^2}\ln\left(\frac{2N}{p}\right)$$

The same result is used in [34]. In both cases, the desired samples are proportional to a constant measuring the size of the problem data and inversely proportional to the desired tolerance squared, as in our case.

In [53], high probability results were derived with the accuracy of the function gradient and Hessian estimates δ_g and δ_H being bounded by ϵ. In part because of the accuracy requirements on the gradient, the resulting sampling bounds are $\mathcal{O}(\epsilon^{-2})$. In our setting, we require the function accuracy to be of order $\epsilon^{3/2}$, where ϵ is the gradient tolerance, yet the dependencies on ϵ suggested by Lemma 3.1 and 4.4 are similar to those in the literature [50, 53]: they are even identical in terms of gradient and Hessian sampling costs. As mentioned above, the sampling rate conditions are closely related to the norm of the steps. This connection was used to define probabilistic models in trust-region methods [3]. A version of our algorithm that would assume access to exact function values but subsampling derivatives would have the same sample complexity as the aforementioned methods [50, 53]: our analysis and discussion regarding stationarity would remain relevant in this context, because of our use of inexact derivatives.

We observe that the sample size, as being proportional to the square of the problem bounds, should be proportional to n^2, and furthermore the upper bounds U_g on the gradients can also grow with n. Thus, unless the setting is a regime with a very large number of data samples and a relatively small number of variables, the computed required sampling size could approach N. It appears that this is an issue associated with the required worst-case sampling rates for higher-order sampling algorithms in general, given the literature described above. To the best of our knowledge, tightening these bounds in a general setting remains an open area of research, even though improved sample complexities can be obtained by exploiting various specific problem structures.

5 Numerical experiments

In this section, we present a numerical study of an implementation of our proposed framework on several machine learning tasks involving both real and simulated data. Our goal is to advocate for the use of second-order methods, such as the one described in this paper, for certain training problems with a nonconvex optimization landscape. We are particularly interested in highlighting notable advantages of going beyond first-order algorithms, which are still the preferred approaches in most machine learning problems. Indeed, first-order methods have lower computational complexity than second-order methods (that is, they require less work to compute a step at each iteration), but second-order methods possess better iteration complexity than first-order ones (they converge in fewer iterations). In many machine learning settings, the lower computational complexity has more significance than the iteration complexity, and therefore first-order algorithms are viewed as superior. Nevertheless, our results suggest that second-order schemes can be competitive in certain contexts.

Our results are meant to be illustrative, and as such, differ from an exhaustive numerical investigation in a number of ways. First, as the number of optimization variables (model parameters in the machine learning model) increases, performing factorization and exact linear algebra solves as required by the algorithm to obtain a Newton step or negative curvature step becomes computationally infeasible. In such settings, iterative linear algebra, i.e. Hessian-free methods using Hessian-vector products, must be used to approximate these directions in practice. On the one hand, the computational complexity of these approaches scale as $O(ns)$, with n being the dimension of the problem, and s the number of Hessian-vector products, as opposed to $O(n^3)$ for full Hessian methods, thus making the step computations more tractable. Several practical approaches have already been proposed [37, 49], yet those are not combined with a stochastic line-search technique such as the one proposed in this paper. As a result, introducing Hessian-free tools would require substantial changes to the convergence analysis of our algorithm, complicating it significantly. We thus focus our experiments on architectures with a relatively small number of network parameters (optimization variables). We point out that there exist important problems where the number of optimization variables n is indeed small compared to the number of samples N. Besides logistic regression and shallow networks, the recent interest in formulating sparser architectures such as `efficientnet` [45] for problems training huge amounts of data, in order to lighten memory loads in computation-heavy and memory-light HPC hardware, indicates the potential for increased applications of this sort in the future.

Besides, we believe that the use of efficient second-order methods based on inexact linear algebra will require a substantial effort of software development within the libraries typically used for deep learning. Our implementation is based on libraries that do have some functionality for computation of second derivatives but with limitations regarding linear algebra operations, and the library we used was the best we could find for this purpose. As a result, our algorithm would likely be disadvantaged if being compared with state-of-the-art methods implemented within standard librairies and finely tuned for standard problems. Thus, even though we are aware that the most popular first-order methods in practice such as `ADAM` [26] often incorporate advanced features, we believe that a comparison with these schemes would be unfair to the second-order algorithm presented in this paper. Still, we believe that there is value in putting our approach in perspective with a simple framework emblematic of the first-order methods used in machine learning. For these reasons, we compare our implementation with a vanilla SGD method using

a constant step size (learning rate). Moreover, we do not perform a completely extensive tuning to obtain an optimal SGD learning rate, but just consider several standard ones as typically chosen a priori. Our goal is to illustrate the sensitivity of the method to such a parameter. By comparison, our second-order method relies on a stochastic line-search technique, that removes the need for heavy hyperparameter tuning. We again emphasize that our experiments aim at indicating the interest of second-order methods vis-à-vis first-order ones, thereby pointing to the desirability of developing tailored libraries for large-scale training problems in the future.

Finally, in all the architectures considered, we require that they result in continuously differentiable objective functions, so as to lie within the assumptions of our convergence analysis. Of course, in practice, activations such as rectified linear units (ReLUs) that introduce nonsmoothness into the objective landscape of training are regularly used to good effect. However, as we are interested in studying the optimization properties of the training specifically, we wish to match the numerical results to the theoretical results in our tests. We point out that many state-of-the-art first-order optimization algorithms do not enjoy convergence theory for nonsmooth nonconvex objectives either, and only standard diminishing stepsize stochastic subgradient methods [30, 20] or ones with a simple momentum term [28] enjoy asymptotic convergence properties. In future work, it will be interesting to see how the second-order methods perform on more standard nonsmooth activations numerically. However, we feel that such a work would only be justified after confirmation of the numerical performance of our framework in theoretically motivated settings.

5.1 Implementation

In our implementation of the `ALAS` method as described in Algorithm 2, we chose the values $\epsilon = 10^{-5}$, $\eta = 10^{-2}$, and $\theta = 0.9$. Note that there are several differences between Algorithm 2 and Algorithm 1. As mentioned in Section 2, Algorithm 1 departs from the method by Royer and Wright [44] in that we did not consider steps along the negative gradient direction, in order to simplify our theoretical analysis. We have re-introduced these steps in our implementation. This change for practical performance has proper intuitive justification: if the negative gradient direction is already a good direction of negative curvature, or the Hessian is flat along its direction, then the negative gradient direction already enjoys second-order properties that ensure it is a good descent direction for the function. Secondly, we modified the condition to select a negative curvature step from $\lambda_k < -\epsilon^{1/2}$ to $\lambda_k < -\|g_k\|^{1/2}$. The former was necessary to handle the case of a stationary point for which $\|g_k^+\| < \epsilon \leq \|g_k\|$ and $\lambda_k > -\epsilon^{1/2}$ in an appropriate fashion in the analysis of Section 3.2, but this case was rarely encountered in practice. Thirdly, we replaced the cubic decrease condition (7) by a sufficient decrease quadratic in the norm of the step, see (55) in Algorithm 2. This is again for performance reasons, and although theoretical results could be established using this condition, it does not appear possible to obtain the same dependencies in ϵ as given in Theorem 4.3. We recall that the cubic decrease condition has been instrumental in deriving optimal iteration complexity bounds for Newton-type methods [14, 44].

The (modified) ALAS framework and a standard stochastic gradient descent (SGD) algorithm were both implemented in Python 3.7. Various Python libraries were used, including JAX [11] for efficient compilation and automatic differentiation, as well as NumPy [48], SciPy [25] and Pandas [36]. All our experiments were run using Intel Xeon CPU at 2.30 GHz and 25.51 GB of RAM, using a Linux distribution Ubuntu 18.04.3 LTS with Linux kernel 4.14.137.

Algorithm 2: ALAS, as implemented.

Initialization: Choose $x_0 \in \mathbb{R}^n$, $\theta \in (0,1), \eta > 0$, $\epsilon > 0$.
for $k = 0, 1, ...$ **do**

1. Draw a random sample set $\mathcal{S}_k \subset \{1, \ldots, N\}$, and compute the associated quantities $g_k := g(x_k; \mathcal{S}_k), H_k := H(x_k; \mathcal{S}_k)$. Form the model:
$$m_k(x_k + s) := m(x_k + s; \mathcal{S}_k). \tag{51}$$
2. Compute $R_k = \frac{g_k^T H_k g_k}{\|g_k\|^2}$.
3. If $R_k < -\|g_k\|^{1/2}$ then set $d_k = \frac{R_k}{\|g_k\|} g_k$ and go to the Line-search step.
4. Else if $R_k < \|g_k\|^{1/2}$ and $\|g_k\| \geq \epsilon$ then set $d_k = -\frac{g_k}{\|g_k\|^{1/2}}$ and go to the Line-search step.
5. Compute λ_k as the minimum eigenvalue of the Hessian estimate H_k.
If $\lambda_k \geq -\epsilon^{1/2}$ and $\|g_k\| = 0$ set $\alpha_k = 0,\ d_k = 0$ and go to Step 10.
6. If $\lambda_k < -\|g_k\|^{1/2}$, Compute a negative eigenvector v_k such that
$$H_k v_k = \lambda_k v_k,\ \|v_k\| = -\lambda_k,\ v_k^\top g_k \leq 0, \tag{52}$$
set $d_k = v_k$ and go to the line-search step.
7. If $\lambda_k > \|g_k\|^{1/2}$, compute a Newton direction d_k solution of
$$H_k d_k = -g_k, \tag{53}$$
go to the line-search step.
8. If d_k has not yet been chosen, compute it as a regularized Newton direction, solution of
$$\left(H_k + (\|g_k\|^{1/2} + \epsilon^{1/2})\mathbb{I}_n\right) d_k = -g_k, \tag{54}$$
and go to the line-search step.
9. **Line-search step** Compute the minimum index j_k such that the step length $\alpha_k := \theta^{j_k}$ satisfies the decrease condition:
$$m_k(x_k + \alpha_k d_k) - m_k(x_k) \ \leq\ -\frac{\eta}{2}\alpha_k^2 \|d_k\|^2. \tag{55}$$
10. Set $x_{k+1} = x_k + \alpha_k d_k$.
11. Set $k = k + 1$.

end

5.2 Classification on the IJCNN1 dataset

We first tested our algorithms on a binary classification task for the IJCNN1 dataset (with $N = 49,990$ samples and $n = 22$ features per sample). Our goal was to train a neural network for binary classification. We used two different architectures: one had one hidden layer with 4 neurons, the other had two hidden layers with 4 neurons each. Both networks used the hyperbolic tangent activation function $\phi(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$ and mean-squared error (MSE) loss, resulting in a twice continuously differentiable optimization problem, with a highly nonlinear objective. The results we present were obtained using samples corresponding to 20 %, 10 %, 5% and 1% of the dataset, with a runtime of 10 seconds. The dataset was partitioned into disjoint samples of given sizes for each data pass (epoch) and these samples were randomly repartitioned every data pass. The MSE loss is defined as

$$l(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (y_i - \widehat{y_i})^2 \,, \tag{56}$$

where $\mathcal{B}$ is the sampled dataset (mini-batch), $|\mathcal{B}|$ the size of the mini-batch, $y_i \in \{-1, 1\}$ the actual binary label of the sample i and $\widehat{y_i}$ is the predicted label of for the sample i.

We tested SGD with several possible values for the learning rate, namely 1, 0.6, 0.3, 0.1, 0.01, 0.001. Figure 1 shows the performance of a subset of those values, and as one may expect, SGD is quite sensitive to the choice of the step size. For this reason, we compare ALAS with the SGD variation corresponding to the best step size value for this problem. Note that the step size in ALAS is selected adaptively through a line search, which makes this method more robust.

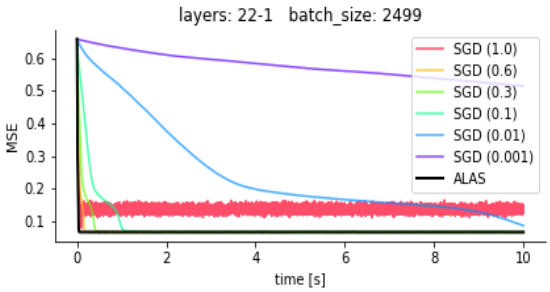
Figures 2 and 3 compare ALAS and SGD with the classical learning rate value for this task (0.01) on the two networks with one and two hidden layers, respectively. The results are summarized in Table 1, where we compare ALAS with the best SGD variant and the standard variant with a learning rate of 0.01. To mitigate the iteration and overall cost of both algorithms, we compared the methods based on the minimal error reached over the last 10 seconds of the run and on the median loss over the last two seconds. In general, ALAS outperforms SGD on the IJCNN1 task in terms of these metrics.

The distribution of the chosen step types as described in Algorithm 2 for a few selected runs for the IJCNN1 task is shown in Figure 4. The step type distribution is generally very different for different architectures but some common patterns were observed. The most frequently used step type is the *Regularized Newton*.

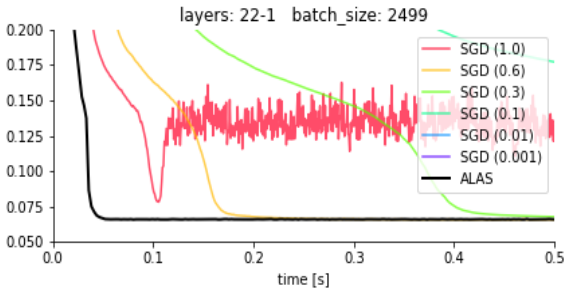
5.3 Experiment using the A9A dataset

The second set of experiments was run on the A9A training dataset, also part of the LIBSVM library [16]: for this classification dataset, we have $N = 32,561$ samples, each of them possessing $n = 123$ features.

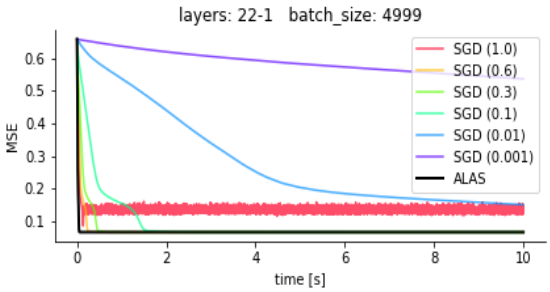
For this task, we trained two neural networks without hidden layers and with one hidden layer having a single neuron; the optimized function was the MSE as in the first experiment. Our batch sizes consisted of 100%, 20%, 10% and 5% of the dataset size, and the mean-squared error loss was used. We tested four variants of SGD corresponding to the values 1, 0.6, 0.3, 0.1 for the learning rate, and selected the best variant for comparison with ALAS. The results are shown in Figures 6 and 7, as well as Table 2.



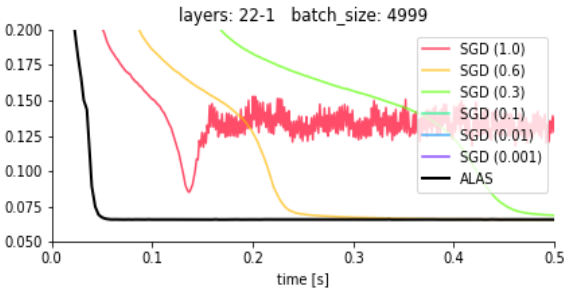
(a) 22-1 5%



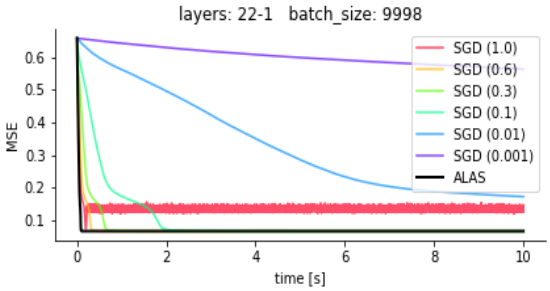
(b) 22-1 5% (zoom)



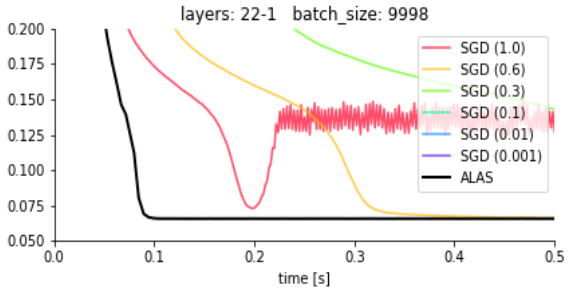
(c) 22-1 10%



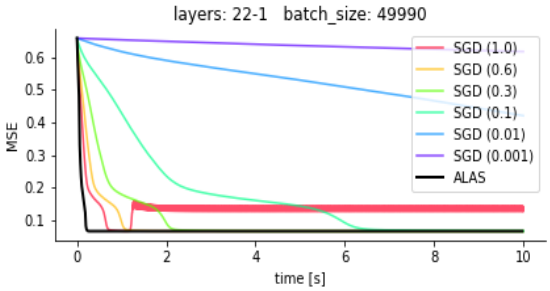
(d) 22-1 10% (zoom)



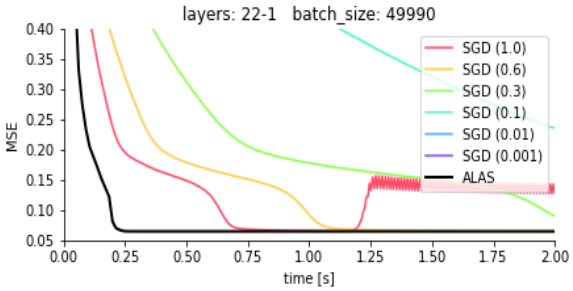
(e) 22-1 20%



(f) 22-1 20% (zoom)

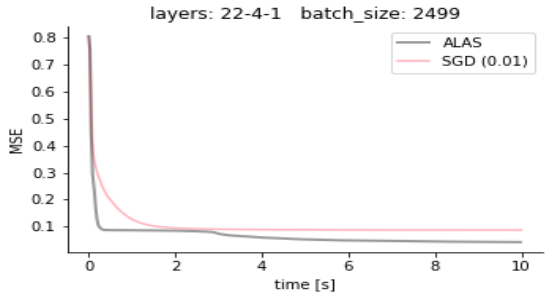


(g) 22-1 100%

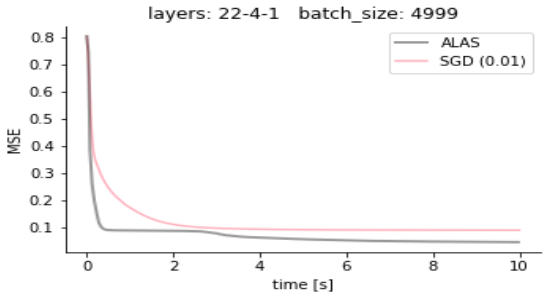


(h) 22-1 100% (zoom)

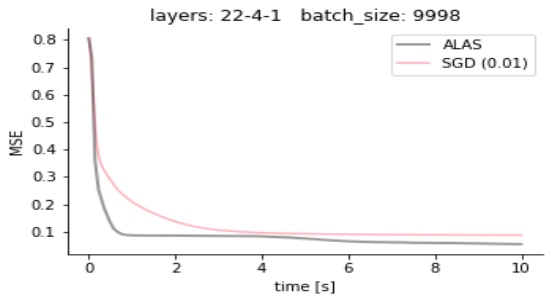
Figure 1: Evaluation of the ALAS algorithm on the IJCNN1 dataset with a simple neural network with 22 input neurons, no hidden layer and an output neuron (22-1) compared to the SGD with various learning rates. Full losses are depicted.



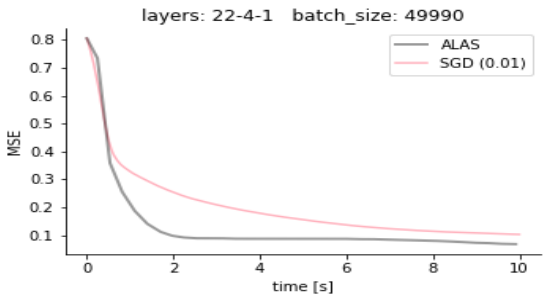
(a) 22-4-1 5%



(b) 22-4-1 10%

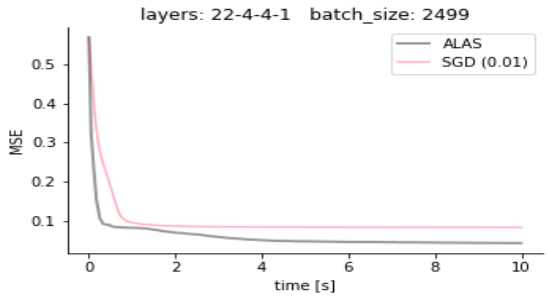


(c) 22-4-1 20%

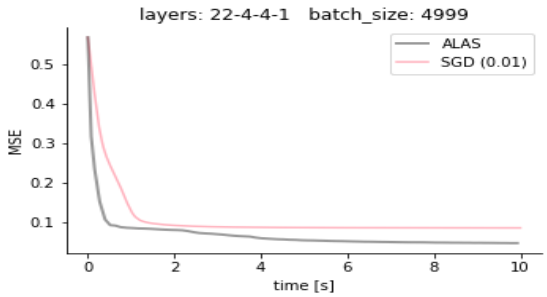


(d) 22-4-1 100%

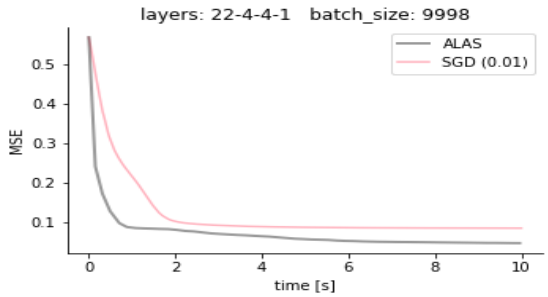
Figure 2: Evaluation of the ALAS algorithm on the IJCNN1 dataset with a simple neural network with 22 input neurons, 4 neurons in the hidden layer and an output neuron (22-4-1) compared to the SGD with a default learning rate 0.01. Full losses are depicted.



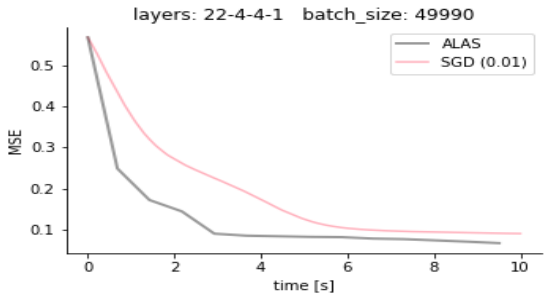
(a) 22-4-4-1 5%



(b) 22-4-4-1 10%

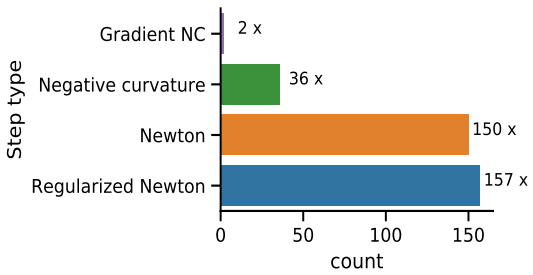


(c) 22-4-4-1 20%

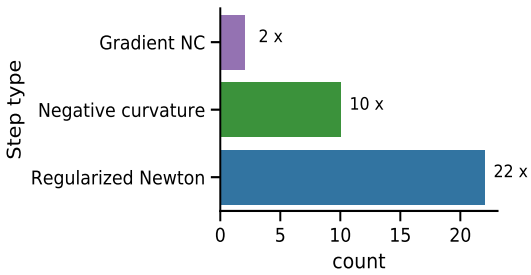


(d) 22-4-4-1 100%

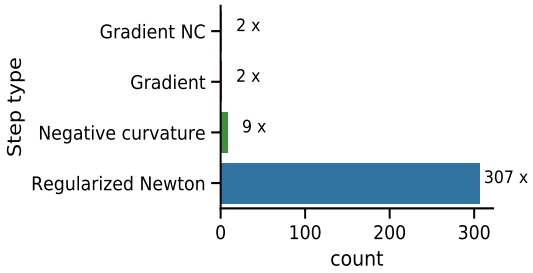
Figure 3: Evaluation of the ALAS algorithm on the IJCNN1 dataset with a simple neural network with 22 input neurons, two hidden layers each with 4 neurons and an output neuron (22-4-4-1) compared to the SGD with a default learning rate 0.01. Full losses are depicted.



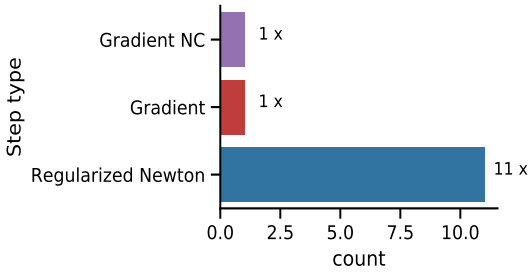
(a) 22-1 100%



(b) 22-4-1 100%

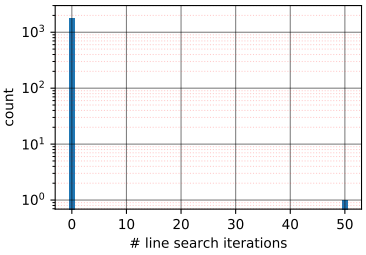


(c) 22-4-4-1 1%

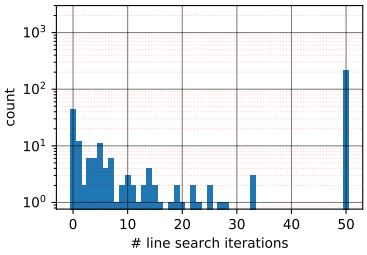


(d) 22-4-4-1 100%

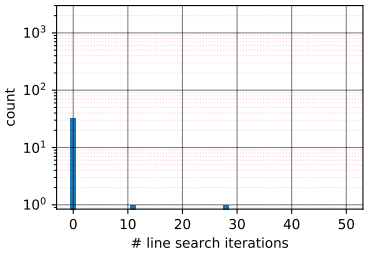
Figure 4: The step type distribution of a single run of ALAS algorithm on the IJCNN1 task for different architectures and sampling sizes.



(a) 22-1 20%



(b) 22-1 100%



(c) 22-4-1 100%

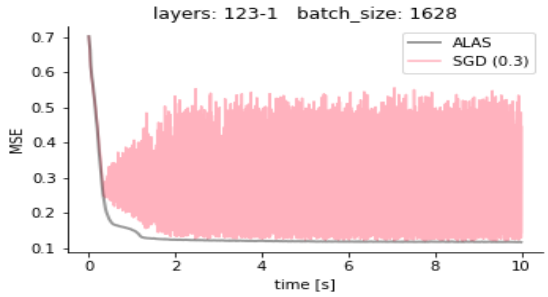
Figure 5: Plots of the number of line-search iterations during each update for the IJCNN1 task for selected runs of the ALAS algorithm with different architectures and sampling sizes. The maximum number of line-search iterations was set to 50.

layers	alg.	π_k	min loss	median loss [8-10]s	iter.
22-4-1	ALAS	1%	**0.0463**	**0.0478**	551
22-4-1	SGD (0.3)	1%	0.0528	0.0558	12033
22-4-1	SGD (0.01)	1%	0.0872	0.0873	11779
22-4-1	ALAS	5%	**0.0449**	**0.0462**	238
22-4-1	SGD (0.6)	5%	0.0514	0.0569	8439
22-4-1	SGD (0.01)	5%	0.0875	0.0876	8471
22-4-1	ALAS	10%	**0.0492**	**0.0510**	124
22-4-1	SGD (0.6)	10%	0.0557	0.0626	6293
22-4-1	SGD (0.01)	10%	0.0880	0.0883	5773
22-4-1	ALAS	20%	0.0611	**0.0628**	76
22-4-1	SGD (0.6)	20%	**0.0598**	0.0654	4389
22-4-1	SGD (0.01)	20%	0.0887	0.0891	4443
22-4-4-1	ALAS	1%	**0.0434**	**0.0451**	320
22-4-4-1	SGD (0.3)	1%	0.0438	0.0454	11121
22-4-4-1	SGD (0.01)	1%	0.0841	0.0842	10997
22-4-4-1	ALAS	5%	**0.0450**	**0.0457**	114
22-4-4-1	SGD (0.3)	5%	0.0468	0.0500	7894
22-4-4-1	SGD (0.01)	5%	0.0845	0.0846	7792
22-4-4-1	ALAS	10%	**0.0470**	**0.0477**	68
22-4-4-1	SGD (0.3)	10%	0.0519	0.0571	5165
22-4-4-1	SGD (0.01)	10%	0.0849	0.0850	5191
22-4-4-1	ALAS	20%	**0.0491**	**0.0498**	42
22-4-4-1	SGD (0.3)	20%	0.0545	0.0587	3728
22-4-4-1	SGD (0.01)	20%	0.0852	0.0854	3694

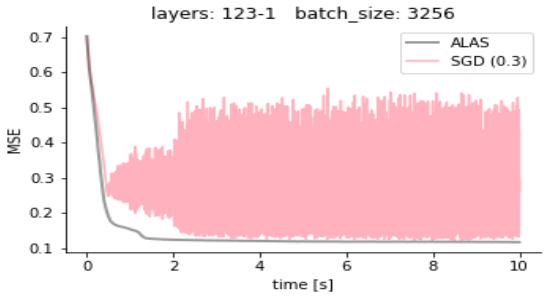
Table 1: Minimal (full) losses reached over the given time period $t = 10$ s on the IJCNN1 task. The number in the parenthesis for SGD entries is the learning rate. The column *median loss [8-10]s* shows the median loss over the last two seconds of the run.

The distribution of the individual type of steps as described in Algorithm 2 for selected runs for the A9A task is shown in Figure 8. As in the case of IJCNN1 task, the most frequently used step type is the *Regularized Newton*. We note that when a full sample is used, *Negative curvature* steps are more common, suggesting that the problem is quite nonconvex. Note that in all of our runs (including those not reported here), other step choices were used less than 10 times. The distributions of number of line-search iterations for several runs of the ALAS algorithm are shown in Figure 9.

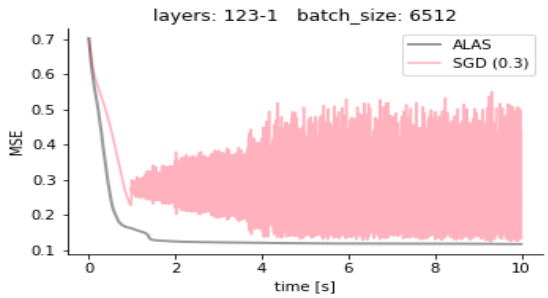
Our method, ALAS, was significantly better for all sampling sizes on training the first network. However, for the second network with more optimization variables (learning parameters), ALAS appears to slow down relative to SGD. This appears due to our use of exact linear algebra techniques: as explained in the introduction of this section, the results could be better for an inexact variant of ALAS.



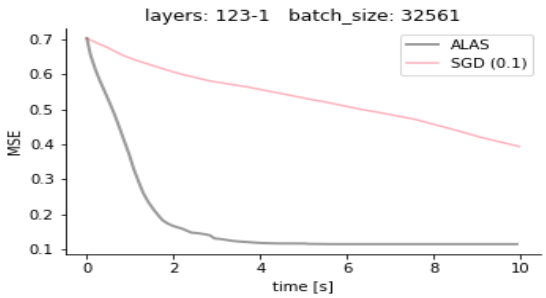
(a) 123-1 5%



(b) 123-1 10%

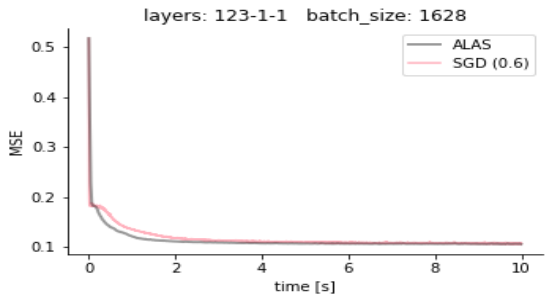


(c) 123-1 20%

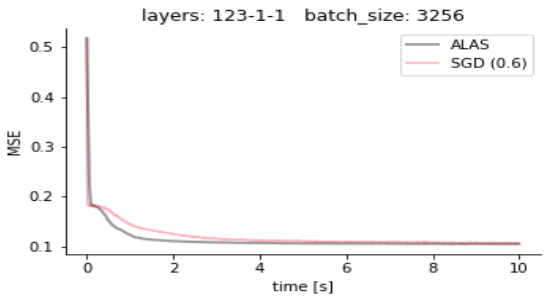


(d) 123-1 100%

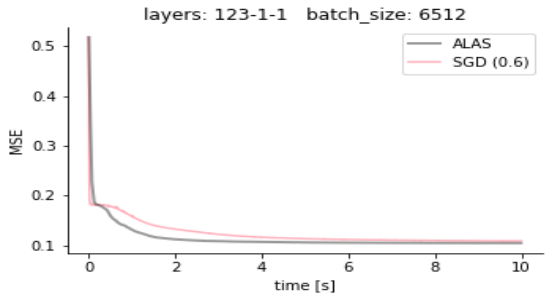
Figure 6: Evaluation of the ALAS algorithm on the A9A dataset with a simple neural network with 123 input neurons, no hidden layer and an output neuron (123-1) compared to the SGD with best performing learning rate. Full losses are depicted.



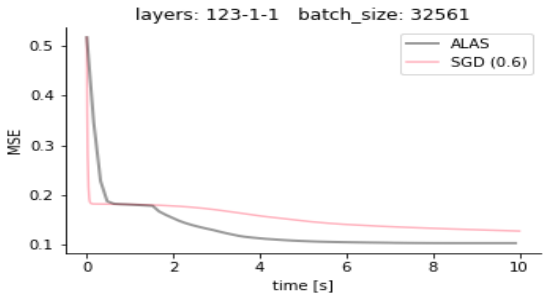
(a) 123-1-1 5%



(b) 123-1-1 10%

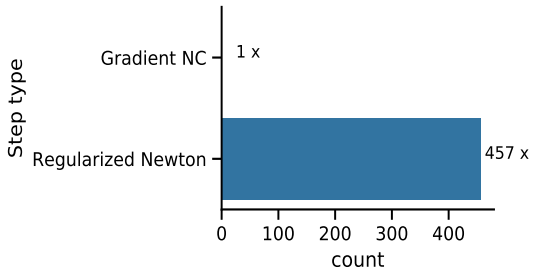


(c) 123-1-1 20%

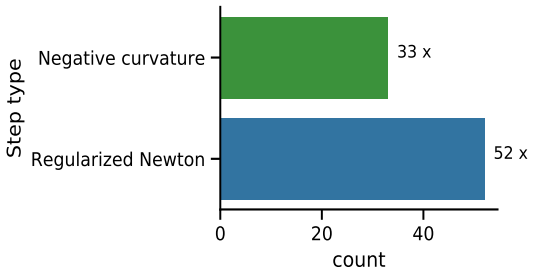


(d) 123-1-1 100%

Figure 7: Evaluation of the ALAS algorithm on the A9A dataset with a simple neural network with 123 input neurons, hidden layer with a single neuron and an output neuron (123-1-1) compared to the SGD with best performing learning rate. Full losses are depicted.



(a) 123-1 5%



(b) 123-1 100%

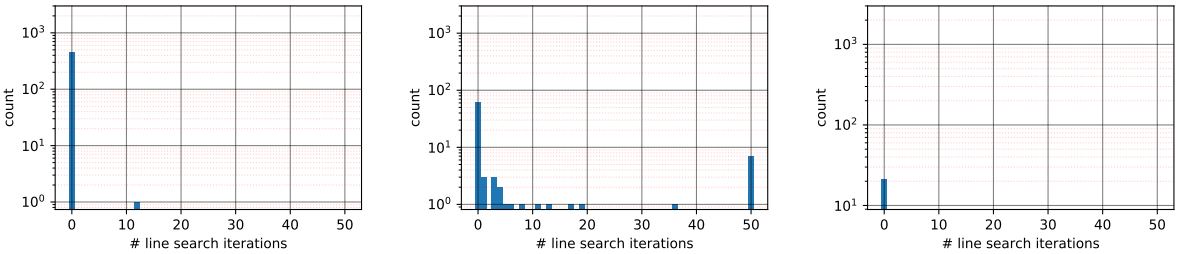
Figure 8: The step type distribution of a single run of ALAS algorithm on the A9A task for the 123-1 architecture and two sampling sizes.

layers	alg.	π_k	min loss	median loss [8-10]s	iter.
123-1	ALAS	5%	**0.1176**	**0.1182**	458
123-1	SGD (0.1)	5%	0.1224	0.1231	8260
123-1	ALAS	10 %	**0.1168**	**0.1176**	360
123-1	SGD (0.1)	10 %	0.1245	0.1255	6168
123-1	ALAS	20 %	**0.1163**	**0.1171**	255
123-1	SGD (0.1)	20 %	0.1396	0.1482	4152
123-1	ALAS	100 %	**0.1151**	**0.1151**	85
123-1	SGD (0.3)	100 %	0.2157	0.2786	645
123-1-1	ALAS	5 %	**0.1051**	**0.1058**	433
123-1-1	SGD (0.6)	5 %	0.1059	0.1064	8449
123-1-1	ALAS	10 %	**0.1047**	**0.1054**	292
123-1-1	SGD (0.6)	10 %	0.1070	0.1076	6165
123-1-1	ALAS	20 %	**0.1049**	**0.1051**	213
123-1-1	SGD (0.6)	20 %	0.1090	0.1095	4111
123-1-1	ALAS	100%	**0.1034**	**0.1035**	64
123-1-1	SGD (0.6)	100 %	0.1277	0.1304	1115
123-2-1	ALAS	5 %	0.1428	0.1437	20
123-2-1	SGD (0.6)	5 %	**0.1066**	**0.1085**	5992
123-2-1	ALAS	10 %	0.1464	0.1482	13
123-2-1	SGD (0.6)	10 %	**0.1094**	**0.1129**	3948
123-2-1	ALAS	20 %	0.1436	0.1452	14
123-2-1	SGD (0.6)	20%	**0.1163**	**0.1213**	2449
123-2-1	ALAS	100 %	0.1550	0.1579	6
123-2-1	SGD (0.6)	100 %	**0.1353**	**0.1365**	819

Table 2: The comparison of the minimal (full) losses reached over the given time period $t = 10\ s$ on the A9A task. The number in the parenthesis for SGD entries is the step size. The column *median loss [8-10]s* shows the median loss over the last two seconds.

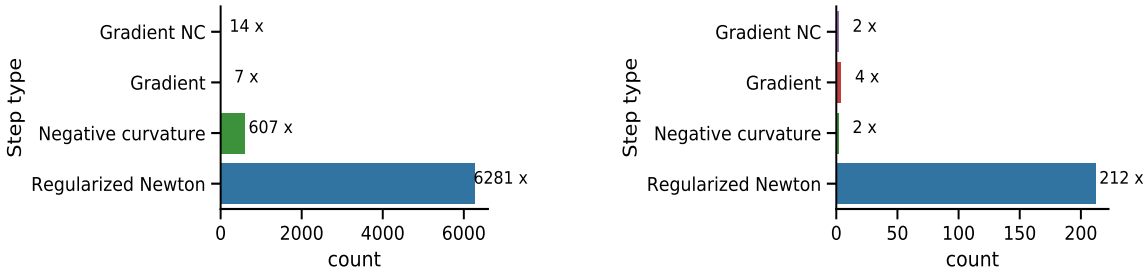
5.4 Transfer learning using MNIST

The current implementation of ALAS is very suitable for transfer learning, where only a few layers of a neural network need to be trained (i.e. only a few parameters need to be optimized). To demonstrate this, we first trained all the parameters in a convolutional neural network (CNN) on a subset of the MNIST dataset [32] (classification of digits 0 - 7). The CNN had 9 convolution layers (with 16 filters for the first layer, 32 filters for the second one, 64 filters for the other layers), followed by a global max pooling layer and a single dense classification layer with softmax activation function and 25% dropout for the eight digits (8 output neurons). Once this network was trained, the classification layer was removed and the extracted features were used for transfer learning to classify the remaining two digits (8 and 9) images of the MNIST dataset, that were unseen during the training of the original CNN. The new classification layers were then trained using the ALAS and the SGD algorithms (we did the tests using the following step sizes 1, 0.6, 0.3, 0.1), with sample sizes corresponding to 100%, 20 %, 10 %, and 5% of the entire data. The total number of samples with the digits 8 and 9 is $N = 11800$ and the input



(a) 123-1 5% (b) 123-1 100% (c) 123-2-1 100%

Figure 9: Plots of the number of line-search iterations during each update for the A9A task for selected runs of the ALAS algorithm with different architectures and sampling sizes. The maximum number of line-search iterations was set to 50.



(a) 8-2-1 5% (b) 8-4-1 100%

Figure 10: The step type distribution of a single run of ALAS algorithm on the transfer learning task.

to the trained layer are 8 neurons. The optimized function was the MSE. As shown in Table 3, ALAS outperformed the SGD variants in most of the runs. Note that when the whole dataset was used, i.e. $\mathcal{S}_k = \{1, \ldots, N\}$ for all k, ALAS outperforms the SGD variants by a significant margin for all four neural network architectures.

The distribution of the individual type of steps as described in Algorithm 2 for selected runs for the transfer learning task is shown in Figure 10. Again, the most frequently used step type was the *Regularized Newton* and the others occurred very rarely. The distributions of number of line-search iterations for selected runs of the ALAS algorithm are shown in Figure 11; the line search was usually rather short (0 iterations of were dominating for most runs) but it was still used quite often.

5.4.1 Transfer learning without pre-computed features

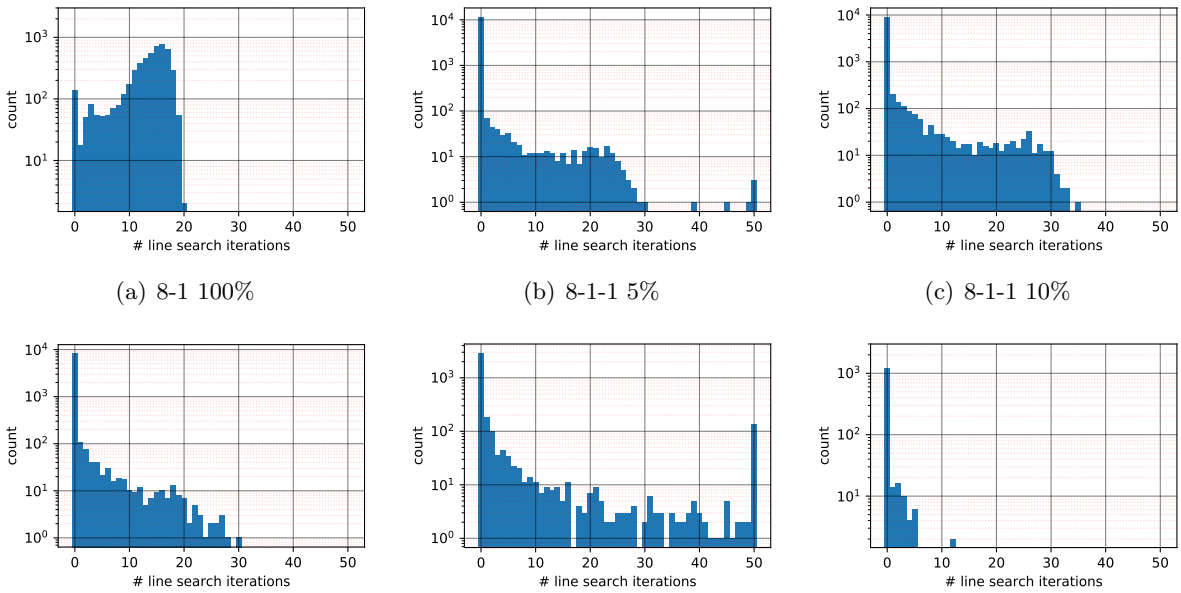
The experiment above is using pre-computed features, i.e. the original images were run through the pre-trained network once and the saved output was used for the optimization of the top layers. While this technique brings massive speed-ups as it is not necessary to compute the outputs for the same images repeatedly, it can only be used when there are repeated images. This prevents the use of online data augmentation, where the individual images are randomly

layers	alg.	π_k	min loss	median loss [8-10]s	iter.
8-1	ALAS	5%	**0.2739**	**0.2752**	12220
8-1	SGD (1.0)	5%	0.2796	0.2801	20264
8-1	ALAS	10%	**0.2732**	**0.2740**	11514
8-1	SGD (1.0)	10%	0.2795	0.2799	20872
8-1	ALAS	20%	**0.2728**	**0.2732**	10691
8-1	SGD (1.0)	20%	0.2796	0.2800	20141
8-1	ALAS	100%	**0.2587**	**0.2592**	5118
8-1	SGD (1.0)	100%	0.2800	0.2803	17832
8-1-1	ALAS	5%	**0.2721**	**0.2815**	11887
8-1-1	SGD (0.3)	5%	0.2796	0.2801	20264
8-1-1	ALAS	10%	**0.2667**	**0.2722**	10116
8-1-1	SGD (0.6)	10%	0.2738	0.2753	20649
8-1-1	ALAS	20%	**0.2657**	**0.2704**	8593
8-1-1	SGD (0.6)	20%	0.2736	0.2749	20327
8-1-1	ALAS	100%	**0.2362**	**0.2363**	3630
8-1-1	SGD (0.3)	100%	0.2751	0.2754	16697
8-2-1	ALAS	5%	**0.2649**	0.2762	6964
8-2-1	SGD (1.0)	5%	0.2667	**0.2702**	20975
8-2-1	ALAS	10%	**0.2632**	**0.2674**	5778
8-2-1	SGD (1.0)	10%	0.2681	0.2705	19502
8-2-1	ALAS	20%	**0.2602**	**0.2625**	3609
8-2-1	SGD (1.0)	20%	0.2695	0.2712	16311
8-2-1	ALAS	100%	**0.2295**	**0.2301**	1151
8-2-1	SGD (1.0)	100%	0.2754	0.2762	7232
8-4-1	ALAS	5%	0.2516	**0.2567**	2245
8-4-1	SGD (1.0)	5%	**0.2524**	0.2550	1356
8-4-1	ALAS	10%	0.2524	**0.2550**	1356
8-4-1	SGD (1.0)	10%	**0.2505**	0.2554	19502
8-4-1	ALAS	20%	0.2529	**0.2542**	811
8-4-1	SGD (1.0)	20%	**0.2504**	0.2555	16211
8-4-1	ALAS	100%	**0.2227**	**0.2247**	220
8-4-1	SGD (1.0)	100%	0.2615	0.2645	6377

Table 3: The comparison of the minimal (full) losses reached over the given time period $t = 10$ s on the transfer learning task. The number in the parenthesis for SGD entries is the step size. The column *median loss [8-10]s* shows the median loss over the last two seconds.

transformed (e.g. rotations, scaling) for each batch and the output from the pre-trained layers has to be recomputed every time.

This is a very favorable scenario for the ALAS algorithm as it usually needs much less updates than the SGD even though each update is more costly in terms of operations — the need to recompute the features every time adds fixed costs to update steps of both SGD and ALAS. We have run the same experiment as above but without the pre-computed features with the time limit 1 hour instead of 10 seconds and with different weight initialization; the results



(a) 8-1 100% (b) 8-1-1 5% (c) 8-1-1 10%

(d) 8-1-1 20% (e) 8-1-1 100% (f) 8-2-1 100%

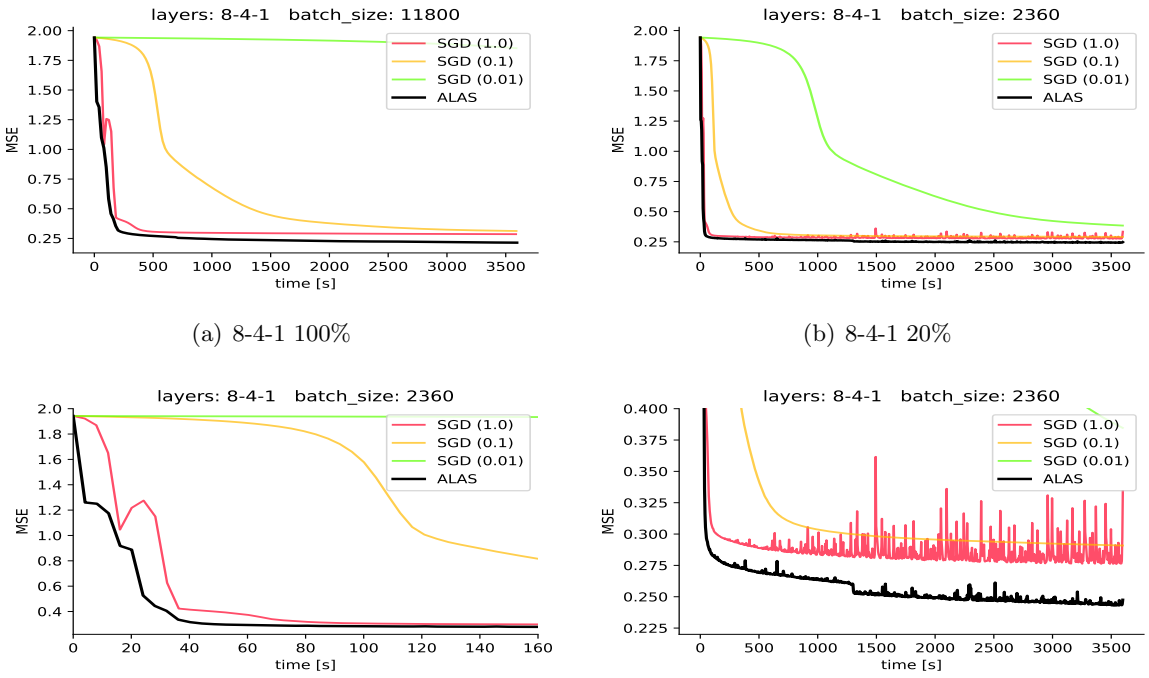
Figure 11: Plots of the number of line-search iterations during each update for the transfer learning task for selected runs of the ALAS algorithm with different architectures and sampling sizes. The maximum number of line-search iterations was set to 50.

are shown in Table 4 and in Fig 12. The number of iterations was similar for both algorithms (only small differences possibly caused by the different utilization of the cloud server) as the feature evaluation step was the dominant for the same *8-4-1* layer configuration for all sample sizes. For higher number of weights, the ALAS performs comparably less iterations than the SGD as the update step is no longer negligible compared to the feature evaluation using the fixed weight neural network (but that is still quite costly and thus we do not observe order of magnitude differences here) as can be seen in the networks with configurations *8-16-4-1* (only ALAS was run to show the small drop in number of iterations) and *8-32-16-1* (the SGD still performs similar number of iterations as for the *8-4-1* layer configuration).

5.5 An artificial dataset

To illustrate the performance of ALAS on highly non-linear problems, we have created two artificial datasets, the first of which (thereafter called NN1) was generated using a neural network with random weights sampled from a normal distribution ($w_i \in \mathcal{N}(0,3)$) and two input neurons, two hidden layers with four and two neurons, respectively. We use hyperbolic tangent activation functions in all layers. To produce the actual training data, 50,000 points were sampled from a uniform distribution ($\vec{x_i} \in \mathcal{R}^2, x_{ij} \in \mathcal{U}(0,1)$) and passed through the generated network to produce target values $\{y_i\}_i$.

The optimization task was then to reconstruct the target values $\{y_i\}_i$ from the generated points $\{\vec{x_i}\}_i$ using different neural networks using the MSE as the loss function. We tried the values 1, 0.6, 0.3, 0.1 for the learning rate of SGD, and we compared these variants with ALAS, using 100%, 20 %, 10 %, and 5% of the sample sizes. The results are shown in Table 5: the



(a) 8-4-1 100%

(b) 8-4-1 20%

(c) 8-4-1 20% (first 160 s)

(d) 8-4-1 20% (detail)

Figure 12: Evaluation of the ALAS algorithm on the transfer learning task without pre-computed features with a simple neural network with 8 input neurons, 4 hidden neurons and an output neuron (8-4-1) compared to the SGD with various learning rates. The Figs 12(c) and 12(d) shows details of the run depicted in Fig 12(b). Full losses are depicted.

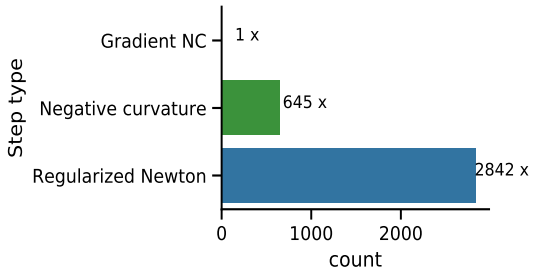
layers	alg.	π_k	min loss	median loss [3400-3600]s	iter.
8-4-1	ALAS	10%	**0.1805**	**0.2483**	1629
8-4-1	SGD (1.0)	10%	0.2003	0.2706	1709
8-4-1	SGD (0.1)	10%	0.2144	0.2853	1683
8-4-1	SGD (0.01)	10%	0.2417	0.3159	1674
8-4-1	ALAS	20%	**0.2034**	**0.2443**	871
8-4-1	SGD (1.0)	20%	0.2356	0.2802	878
8-4-1	SGD (0.1)	20%	0.2488	0.2913	848
8-4-1	SGD (0.01)	20%	0.3657	0.3946	900
8-4-1	ALAS	100%	**0.2142**	**0.2155**	175
8-4-1	SGD (1.0)	100%	0.2858	0.2862	173
8-4-1	SGD (0.1)	100%	0.3126	0.3152	170
8-4-1	SGD (0.01)	100%	1.8525	1.8656	169
8-16-4-1	ALAS	20 %	0.1884	0.2294	748
8-4-1	SGD (1.0)	100%	0.2255	0.2680	870
8-4-1	SGD (0.1)	100%	0.2326	0.2796	869
8-4-1	SGD (0.01)	100%	0.2517	0.3028	878
8-32-16-1	ALAS	20 %	**0.2076**	**0.2664**	562
8-32-16-1	SGD (0.1)	20%	0.2230	0.2709	887
8-32-16-1	SGD (0.01)	20%	0.2480	0.2932	876
8-32-16-1	SGD (1.0)	20%	0.3015	1.9834	886

Table 4: The comparison of the minimal (full) losses reached over the given time period $t = 3600$ s on the transfer learning task without pre-computed features. The number in the parenthesis for SGD entries is the step size. The column *median loss [3400-3600]s* shows the median loss over the last 200 seconds.

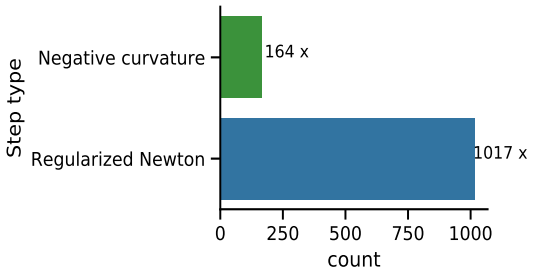
algorithm ALAS performed best with a single exception when it got stuck in a worse local optimum than the SGD variant. The step type distributions for different runs is shown in Figure 13 — the most frequent steps in general for this task were the *Regularized Newton* (strongly dominating) and the *Negative curvature* step.

5.6 A second artificial dataset

The second artificial dataset, called NN2, was created using the same process as described in Section 5.5. For this second dataset, we used a deeper neural network in order to introduce more nonlinearities: this network consisted of four input neurons, three hidden layers with up to four neurons, and one output layer producing the targets $\{y_i\}_i$. The networks used for experiments had one or two hidden layers — first hidden layer had always four neurons and the second had one or two neurons if present. All neurons used a hyperbolic tangent as their activation function, the optimized function was the MSE. Few runs of the ALAS and the SGD algorithms are shown in Figures 14 and 15. The step type distribution for different runs is shown in Figure 16 — the *Regularized Newton* step type was the most frequent. Our implementation of ALAS performed generally better than the best SGD variant, sometimes by a significant margin.

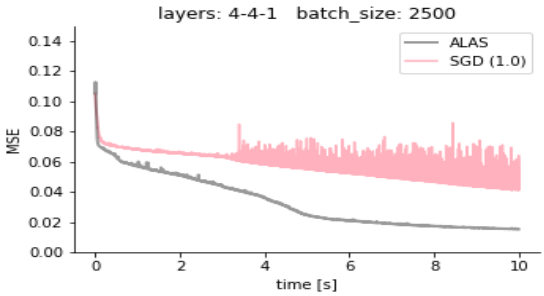


(a) 2-1-1 20%

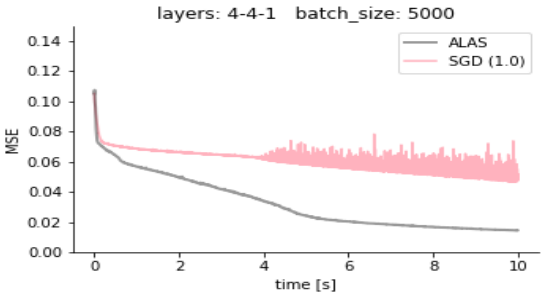


(b) 2-4-2-1 5%

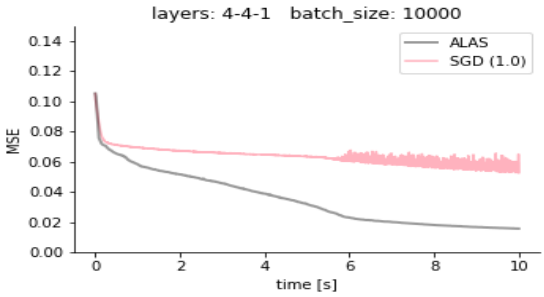
Figure 13: The step type distribution of a single run of ALAS algorithm on the NN1 task.



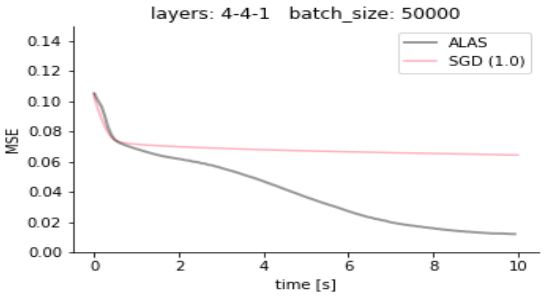
(a) 4-4-1 5%



(b) 4-4-1 10%

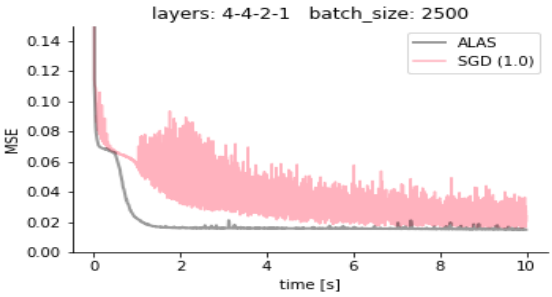


(c) 4-4-1 20%

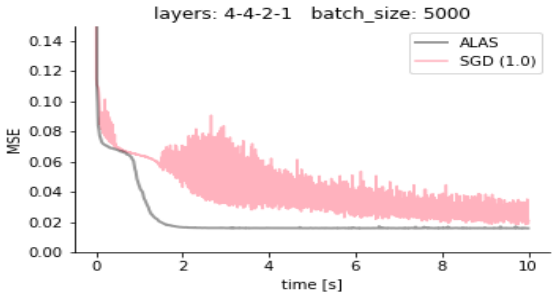


(d) 4-4-1 100%

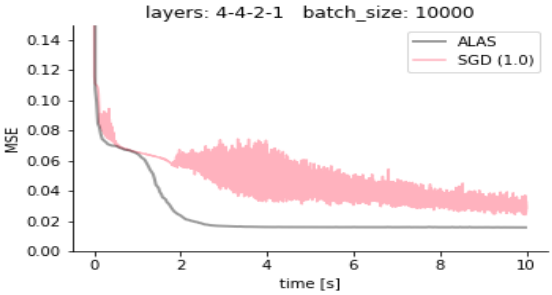
Figure 14: Evaluation of the ALAS algorithm on artifical task NN2 compared to the SGD with best performing learning rate. Full losses are depicted.



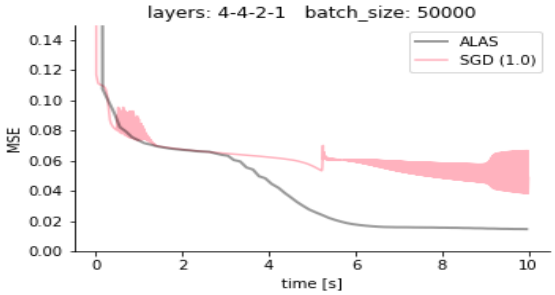
(a) 4-4-2-1 5%



(b) 4-4-2-1 10%

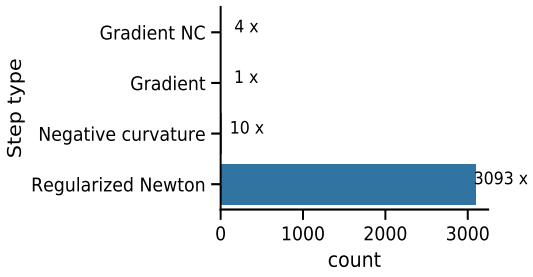


(c) 4-4-2-1 20%

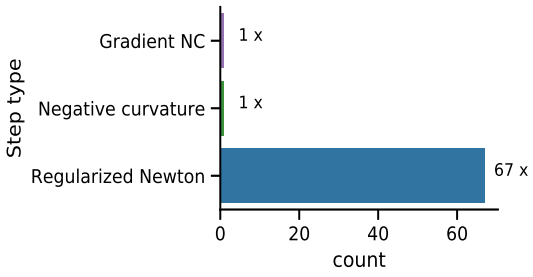


(d) 4-4-2-1 100%

Figure 15: Evaluation of the ALAS algorithm on artificial task NN2 compared to the SGD with best performing learning rate. Full losses are depicted.



(a) 4-2-1 10%



(b) 4-4-2-1 100%

Figure 16: The step type distribution of a single run of ALAS algorithm on the NN2 task.

layers	alg.	π_k	min loss	median loss [8-10]s	iter.
2-1-1	ALAS	5%	$\mathbf{9.080 \times 10^{-10}}$	$\mathbf{1.118 \times 10^{-9}}$	7471
2-1-1	SGD (1.0)	5%	2.492×10^{-6}	2.756×10^{-6}	15432
2-1-1	ALAS	10%	$\mathbf{8.940 \times 10^{-10}}$	$\mathbf{1.014 \times 10^{-9}}$	5595
2-1-1	SGD (1.0)	10%	2.769×10^{-6}	3.094×10^{-6}	14015
2-1-1	ALAS	20%	$\mathbf{8.913 \times 10^{-10}}$	$\mathbf{9.389 \times 10^{-10}}$	3353
2-1-1	SGD (1.0)	20%	3.413×10^{-6}	3.795×10^{-6}	11533
2-1-1	ALAS	100%	$\mathbf{1.150 \times 10^{-9}}$	$\mathbf{1.211 \times 10^{-9}}$	538
2-1-1	SGD (1.0)	100%	7.514×10^{-6}	8.418×10^{-6}	5503
2-2-1	ALAS	5%	$\mathbf{6.586 \times 10^{-10}}$	4.000	5736
2-2-1	SGD (1.0)	5%	1.596×10^{-6}	$\mathbf{1.779 \times 10^{-6}}$	14105
2-2-1	ALAS	10%	$\mathbf{1.568 \times 10^{-10}}$	$\mathbf{2.180 \times 10^{-10}}$	2352
2-2-1	SGD (1.0)	10%	1.859×10^{-6}	2.066×10^{-6}	12134
2-2-1	ALAS	20%	$\mathbf{1.964 \times 10^{-10}}$	$\mathbf{2.230 \times 10^{-10}}$	1267
2-2-1	SGD (1.0)	20%	2.200×10^{-6}	2.441×10^{-6}	10283
2-2-1	ALAS	100%	$\mathbf{9.621 \times 10^{-10}}$	$\mathbf{9.989 \times 10^{-10}}$	304
2-2-1	SGD (1.0)	100%	5.989×10^{-6}	6.683×10^{-6}	3833
2-4-1-1	ALAS	5%	$\mathbf{9.210 \times 10^{-10}}$	$\mathbf{9.232 \times 10^{-10}}$	1529
2-4-1-1	SGD (1.0)	5%	5.474×10^{-7}	5.600×10^{-7}	13224
2-4-1-1	ALAS	10%	$\mathbf{9.942 \times 10^{-10}}$	$\mathbf{1.028 \times 10^{-9}}$	818
2-4-1-1	SGD (1.0)	10%	5.755×10^{-7}	5.875×10^{-7}	10873
2-4-1-1	ALAS	20%	$\mathbf{1.260 \times 10^{-9}}$	$\mathbf{1.336 \times 10^{-9}}$	485
2-4-1-1	SGD (1.0)	20%	6.028×10^{-7}	6.121×10^{-7}	8223
2-4-1-1	ALAS	100%	$\mathbf{4.047 \times 10^{-9}}$	$\mathbf{4.427 \times 10^{-9}}$	94
2-4-1-1	SGD (1.0)	100%	6.759×10^{-7}	6.799×10^{-7}	2819
2-4-2-1	ALAS	5%	$\mathbf{9.266 \times 10^{-10}}$	$\mathbf{9.335 \times 10^{-10}}$	922
2-4-2-1	SGD (1.0)	5%	1.680×10^{-6}	1.864×10^{-6}	11909
2-4-2-1	ALAS	10%	$\mathbf{1.025 \times 10^{-9}}$	$\mathbf{1.053 \times 10^{-9}}$	493
2-4-2-1	SGD (1.0)	10%	2.171×10^{-6}	2.394×10^{-6}	9192
2-4-2-1	ALAS	20%	$\mathbf{1.249 \times 10^{-9}}$	$\mathbf{1.295 \times 10^{-9}}$	262
2-4-2-1	SGD (1.0)	20%	2.907×10^{-6}	3.198×10^{-6}	6850
2-4-2-1	ALAS	100%	$\mathbf{3.590 \times 10^{-9}}$	$\mathbf{3.957 \times 10^{-9}}$	55
2-4-2-1	SGD (1.0)	100%	9.763×10^{-6}	1.089×10^{-5}	2008

Table 5: The comparison of the minimal (full) losses reached over the given time period $t = 10\ s$ on the artificial dataset NN1. The number in the parenthesis for SGD entries is the step size. The column *median loss [8-10]s* shows the median loss over the last two seconds.

6 Conclusion

In this paper, we presented a line-search method for stochastic optimization, wherein the Hessian, gradient, and function values must be estimated through subsampling and cannot be obtained exactly. Using probabilistically accurate models, we derived a complexity result on the expected number of iterations until an approximate measure of stationary is reached for the current model. This result in expectation is complementary to those holding with high probability, i.e.

layers	alg.	π_k	min loss	median loss [8-10]s	iter.
4-4-1	ALAS	5%	**0.0153**	**0.0163**	1654
4-4-1	SGD (1.0)	5%	0.0411	0.0466	13185
4-4-1	ALAS	10%	**0.0145**	**0.0156**	1218
4-4-1	SGD (1.0)	10%	0.0466	0.0513	10548
4-4-1	ALAS	20%	**0.0158**	**0.0168**	727
4-4-1	SGD (1.0)	20%	0.0527	0.0562	7617
4-4-1	ALAS	100%	**0.0122**	**0.0133**	126
4-4-1	SGD (1.0)	100%	0.0646	0.0651	3068
4-4-1-1	ALAS	5%	**0.0515**	0**.0518**	1506
4-4-1-1	SGD (1.0)	5%	0.0523	0.0525	12811
4-4-1-1	ALAS	10%	**0.0515**	**0.0516**	1074
4-4-1-1	SGD (1.0)	10%	0.0525	0.0527	10390
4-4-1-1	ALAS	20%	**0.0516**	**0.0517**	564
4-4-1-1	SGD (1.0)	20%	0.0530	0.0532	7259
4-4-1-1	ALAS	100%	**0.0511**	**0.0511**	106
4-4-1-1	SGD (1.0)	100%	0.0553	0.0557	2814
4-4-2-1	ALAS	5%	**0.0150**	**0.0154**	1046
4-4-2-1	SGD (1.0)	5%	0.0167	0.0222	11814
4-4-2-1	ALAS	10%	**0.0160**	**0.0161**	650
4-4-2-1	SGD (1.0)	10%	0.0185	0.0260	8764
4-4-2-1	ALAS	20%	**0.0159**	**0.0160**	320
4-4-2-1	SGD (1.0)	20%	0.0242	0.0317	6123
4-4-2-1	ALAS	100%	**0.0148**	**0.0153**	69
4-4-2-1	SGD (1.0)	100%	0.0384	0.0489	2222

Table 6: The comparison of the minimal (full) losses reached over the given time period $t = 10$ s on the artificial dataset NN2. The number in the parenthesis for SGD entries is the step size. The column *median loss [8-10]s* shows the median loss over the last two seconds.

with probability of drawing an accurate sample at every iteration. In our setting, we are able to obtain convergence in expectation at a rate which is commensurate with the state of the art, regardless of the presence of outliers in sample estimates resulting in poor steps. We also proposed a practical strategy to assess whether the current iterate is close to a sample point for the original objective, that does not require the computation of the full function. Our numerical experiments showed the potential of the proposed approach on several machine learning tasks, including transfer learning.

We believe that the results of the paper will encourage further study into the design of second-order algorithms despite the prevailing paradigm of using first-order methods for their computationally cheap iterations. In particular, our approach could be helpful in generalizing other line-search techniques to the context of subsampled function values. Moreover, our theoretical analysis, that captures the worst-case behavior of the problem, can serve as a basis for a refined version of our algorithm for structured problems. Developing specialized versions of our algorithm thus represents an interesting avenue of research. In fact, our line-search tools could also serve the development of (stochastic) first-order methods, in which the iterates are

typically subject to a lot of noise, and both learning rate scheduling and tuning are required in order to prevent the training from slowing down. Using a line search allows for a lower training error in expectation, while preventing extensive iterate noise.

Finally, incorporating more practical features such as matrix-free operations and iterative linear algebra would be a natural enhancement of our algorithm, that would follow previous work on second-order methods for neural networks [29]. This could not only have a significant impact on the practical performance, but could also lead to more precise computational complexity results, similarly to what has been observed in deterministic nonconvex optimization. In time, higher-order methods might be made competitive with first-order ones in practice. This will however require considerable software development and interfacing with current popular deep learning libraries. We hope that this paper indicates the value of such an undertaking.

Acknowledgments

We are indebted to Courtney Paquette and Katya Scheinberg for raising an issue with the first version of this paper, that lead to significant improvement of its results.

References

[1] Naman Agarwal, Zeyuan Allen-Zhu, Brian Bullins, Elad Hazan, and Tengyu Ma. Finding approximate local minima faster than gradient descent. In *Annual ACM SIGACT Symposium on Theory of Computing*, pages 1195–1199, 2017.

[2] Zeyuan Allen-Zhu. Natasha 2: Faster non-convex optimization than SGD. In *Advances in Neural Information Processing Systems*, 2018.

[3] Afonso S. Bandeira, Katya Scheinberg, and Luis N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim.*, 24:1238–1264, 2014.

[4] Albert S. Berahas, Raghu Bollapragada, and Jorge Nocedal. An investigation of Newton-sketch and subsampled Newton methods. *arXiv preprint arXiv:1705.06211*, 2017.

[5] El Houcine Bergou, Youssef Diouane, Vyacheslav Kungurtsev, and Clément W. Royer. A stochastic Levenberg-Marquardt method using random models with application to data assimilation. *arXiv preprint arXiv:1807.2176*, 2018.

[6] El Houcine Bergou, Serge Gratton, and Luis N. Vicente. Levenberg-Marquardt methods based on probabilistic gradient models and inexact subproblem solution, with application to data assimilation. *SIAM/ASA J. Uncertain. Quantif.*, 4:924–951, 2016.

[7] Jose Blanchet, Coralia Cartis, Matt Menickelly, and Katya Scheinberg. Convergence rate analysis of a stochastic trust region method for nonconvex optimization. *INFORMS Journal on Optimization*, 1:92–119, 2019.

[8] Raghu Bollapragada, Richard H. Byrd, and Jorge Nocedal. Exact and inexact subsampled Newton methods in optimization. *IMA J. Numer. Anal.*, 39:545–578, 2019.

[9] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching L-BFGS method for machine learning. In *International Conference on Machine Learning*, pages 620–629, 2018.

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

[11] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. *JAX: composable transformations of Python+NumPy programs*, 2018.

[12] Richard H. Byrd, Gillian M. Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM J. Optim.*, 21:977–995, 2011.

[13] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford. Accelerated methods for non-convex optimization. *SIAM J. Optim.*, 28:1751–1772, 2018.

[14] Coralia Cartis, Nicholas I. M. Gould, and Philippe L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. Part II: worst-case function- and derivative-evaluation complexity. *Math. Program.*, 130:295–319, 2011.

[15] Coralia Cartis and Katya Scheinberg. Global convergence rate analysis of unconstrained optimization methods based on probabilistic models. *Math. Program.*, 169:337–375, 2018.

[16] Chih-Chung Chang and Chih-Jen Lin. LIBSVM. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, April 2011.

[17] Ruobing Chen, Matt Menickelly, and Katya Scheinberg. Stochastic optimization using a trust-region method and random models. *Math. Program.*, 169:447–487, 2018.

[18] Frank E. Curtis, Katya Scheinberg, and Rui Shi. A stochastic trust-region algorithm based on careful step normalization. *INFORMS Journal on Optimization*, 1:200–220, 2019.

[19] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

[20] Damek Davis, Dmitriy Drusvyatskiy, Sham Kakade, and Jason D. Lee. Stochastic subgradient method converges on tame functions. *Foundations of Computational Mathematics*, pages 1–36, 2018.

[21] Rick Durrett. *Probability: Theory and Examples*. Camb. Ser. Stat. Prob. Math. Cambridge University Press, Cambridge, fourth edition, 2010.

[22] Murat A Erdogdu and Andrea Montanari. Convergence rates of sub-sampled Newton methods. In *Advances in Neural Information Processing Systems*, pages 3052–3060, 2015.

[23] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM J. Optim.*, 23:2341–2368, 2013.

[24] Serge Gratton, Clément W. Royer, Luis N. Vicente, and Zaikun Zhang. Complexity and global rates of trust-region methods based on probabilistic models. *IMA J. Numer. Anal.*, 38:1579–1597, 2018.

[25] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-05-12].

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[27] Jonas Moritz Kohler and Aurélien Lucchi. Sub-sampled cubic regularization for non-convex optimization. In *International Conference on Machine Learning*, pages 1895–1904, 2017.

[28] Vyacheslav Kungurtsev, Malcolm Egan, Bapi Chatterjee, and Dan Alistarh. Asynchronous stochastic subgradient methods for general nonsmooth nonconvex optimization. *arXiv preprint arXiv:1905.11845*, 2019.

[29] Vyacheslav Kungurtsev and Tomas Pevny. Algorithms for solving optimization problems arising from deep neural net models: smooth problems. *Cisco-CTU WP5 Technical Report*, 2016. Originally proprietary industrial report. Currently available online at https://arxiv.org/abs/1807.00172.

[30] Harold Kushner and G. George Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

[31] Jeffrey Larson and Stephen C. Billups. Stochastic derivative-free optimization using a trust region framework. *Comput. Optim. Appl.*, 64:619–645, 2016.

[32] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[33] Jason D. Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. First-order methods almost always avoid saddle points. *Math. Program.*, 176:311–337, 2019.

[34] Mingrui Liu and Tianbao Yang. On noisy negative curvature descent: Competing with gradient descent for faster non-convex optimization. *arXiv preprint arXiv:1709.08571*, 2017.

[35] Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *J. Mach. Learn. Res.*, 18:1–59, 2017.

[36] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[37] Thomas O'Leary-Roseberry, Nick Alger, and Omar Ghattas. Inexact Newton methods for stochastic nonconvex optimization with applications to neural network training. *arXiv preprint arXiv:1905.06738*, 2019.

[38] Courtney Paquette and Katya Scheinberg. A stochastic line search method with convergence rate analysis. *arXiv preprint arXiv:1807.07994*, 2018.

[39] Mert Pilanci and Martin J. Wainwright. Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence. *SIAM J. Optim.*, 27:205–245, 2017.

[40] Herbert Robbins and David Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Optimizing methods in statistics*, pages 233–257. Elsevier, 1971.

[41] Farbod Roosta-Khorasani and Michael W. Mahoney. Sub-sampled Newton methods I: Globally convergent algorithms. *arXiv preprint arXiv:1601.04737*, 2016.

[42] Farbod Roosta-Khorasani and Michael W. Mahoney. Sub-sampled Newton methods. *Math. Program.*, 174:293–326, 2019.

[43] Sheldon M. Ross. *Stochastic processes*. Wiley, New York, 1996.

[44] Clément W. Royer and Stephen J. Wright. Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization. *SIAM J. Optim.*, 28:1448–1477, 2018.

[45] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[46] Nilesh Tripuraneni, Mitchell Stern, Chi Jin, Jeffrey Regier, and Michael I. Jordan. Stochastic cubic regularization for fast nonconvex optimization. In *Advances in Neural Information Processing Systems*, 2018.

[47] Joel A. Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends© in Machine Learning*, 8:1–230, 2015.

[48] Stefan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 3 2011.

[49] Peng Xu, Farbod Roosta-Khorasani, and Michael W. Mahoney. Second-order optimization for non-convex machine learning: An empirical study. *arXiv preprint arXiv:1708.07827v2*, 2018.

[50] Peng Xu, Farbod Roosta-Khorasani, and Michael W. Mahoney. Newton-type methods for non-convex optimization under inexact hessian information. *Math. Program.*, 2019 (to appear).

[51] Peng Xu, Jiyan Yang, Farbod Roosta-Khorasani, Christopher Ré, and Michael W. Mahoney. Sub-sampled Newton methods with non-uniform sampling. In *Advances in Neural Information Processing Systems*, 2016.

[52] Yi Xu, Rong Jin, and Tianbao Yang. First-order stochastic algorithms for escaping from saddle points in almost linear time. In *Advances in Neural Information Processing Systems*, 2018.

[53] Zhewei Yao, Peng Xu, Farbod Roosta-Khorasani, and Michael W. Mahoney. Inexact non-convex Newton-type methods. *arXiv preprint arXiv:1802.06925*, 2018.