UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

OPTIMIZATION FOR DATA SCIENCE

FINAL PROJECT

ANALYSIS OF SGD VARIANTS

University of Padua

Department of Mathematics

Academic Year 2021/2022

Beatrice Sofia Bertipaglia	mat. 2054766
Chiara Colato	mat. 2062670
Flavia Gianfrate	mat. 2038501
Alberto Sinigaglia	mat. 2044712

Abstract

In this project we aim to analyze three main algorithms, based on the basic *Stochastic Gradient Descent* (*SGD*) method, in order to understand their theoretical and practical properties: *SARAH*, *SpiderBoost* and *SNVRG*. All these algorithms improve the performance of *SGD* and we show that, at the base of these improvements, there is the reduction of the variance: it reduces the uncertainty introduced by the estimator, allowing the algorithm to do steps closer to the correct one. We also see that, using these techniques, it's possible to reach the minimum point with a better gradient complexity than SGD (i.e. $\mathcal{O}(1/\epsilon^4)$) and, therefore, they are a valid and better solutions.
After a first presentation, all the algorithms are tested following the lines of experiments of Zhe Wang in [7] for the smooth case.

1 Introduction

The proposed project concerns the supervised learning paradigm, by solving optimization problems.
We start by analyzing the theoretical properties of each considered algorithm, in order to identify similarities and differences. Then, we study the behaviour of algorithms on training examples of selected datasets, observing if the theoretical claims are confirmed. During all the project we aim to deal with the finite-sum minimization problem given in the form:

$$min_{x\in R^d}\, f(x) \tag{1}$$

where:

$$f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{2}$$

and f denote the total loss function computed on the training sample data, that we assume to be *smooth*. Throughout the presented paper, we assume that there exists a solution x^* for the above cited problem P.

The algorithms we present and discuss about in this paper are based on the *Stochastic Gradient Descent* (*SGD*) algorithm that, in order to compute the updates, considers a different index for each update, calculated at each iteration of the algorithm. However, results show that the algorithm is slow and its performance are sensitive to the variance of the sample gradients, used for the updates.

In order to overcome these issues, new sophisticated algorithms are introduced to solve optimization problems. They do not require the computation of the whole gradient and try to reduce the variance of the *SGD* estimator, making use of past stochastic gradient information, coming from the previous updates.

2 Loss Functions

In our experiments we consider two smooth problems for the minimization of non-convex objective functions.

The first one is represented by the *logistic regression problem*, with a nonconvex regularizer and cross-entropy loss function, that takes the form:

$$\min_{w\in\mathbb{R}^d} f(w) := -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\frac{1}{1+e^{-w^\top x_i}}\right) \;+\; (1-y_i)\log\left(1-\frac{1}{1+e^{-w^\top x_i}}\right)\right] \;+\; \frac{1}{10}\sum_{j=1}^{d}\frac{w_j^2}{1+w_j^2} \tag{3}$$

where $x_i \in \mathbb{R}^n$ are the features, and $y_i \in \{0,1\}$ are the targets. The j-th partial derivative of this objective function, for all $j = 1, ..., d$, is given by the following formula:

$$\frac{\partial}{\partial w_j} f(w) = \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i e^{-w^\top x_i} + (1-y_i)}{1+e^{-w^\top x_i}}(x_i)_j \;+\; \frac{1}{5}\frac{w_j}{(1+w_j^2)^2} \quad . \tag{4}$$

The second problem is the *robust linear regression* one, with nonconvex loss function, that takes the form:

$$\min_{w\in\mathbb{R}^d} f(w) = \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{(y_i - w^\top x_i)^2}{2} + 1\right) \quad (5)$$

where $x_i \in \mathbb{R}^n$ are the features, and $y_i \in \{0, 1\}$ are the targets. The j-th partial derivative of this objective function, for all $j = 1, ..., d$, is in form:

$$\frac{\partial}{\partial w_j} f(w) = -\frac{1}{n}\sum_{i=1}^{n} \frac{2(y_i - w^\top x_i)}{(y_i - w^\top x_i)^2 + 2}(x_i)_j \quad . \quad (6)$$

3 SARAH Algorithm

The first algorithm that we propose in this document is the *Stochastic Recursive Gradient Algorithm* (*SARAH*).
SARAH uses a double gradient update: it makes use of *past stochastic gradient information*, and, occasionally, of *exact gradient information*, but the key step of this algorithm is the recursive update of the stochastic gradient estimate. We can describe the most important properties of *SARAH* as:

- The iterations are divided into two nested loops. In the inner loop the algorithms computes the stochastic gradient using all the accumulated information reached up to that moment: at each iteration it evaluates two stochastic gradients, the current gradient and the previous one, that, recursively thinking, contains all the information about the past updates.
  In the outer loop, instead, the full gradient update is computed.
- It does not requires the storing of all the past updates and so it save storage costs.
- *SARAH* uses a constant learning rate η.
- Thanks to the structure given to the inner loop, the variance of the steps computed goes to zero, making possible the reduction of the sensibility to variance, and the obtaining of more stable results in practice.

As we said, we can present the *SARAH update*: the recursive updating of the stochastic gradient estimate

$$v_t = \nabla f_{it}(x_t) - \nabla f_{it}(x_{t-1}) + v_{t-1} \quad . \quad (7)$$

The above computed quantity is then used to calculate the iterate update

$$x_{t+1} = x_t - \eta v_t \quad . \quad (8)$$

3.1 SARAH: Convergence Analysis

To proceed with the analysis of this proposed algorithm, we have to take in account some known results. In particular we know that *SARAH* algorithm has a linear convergence rate when it's applied on strongly convex optimization problems, while it reaches a sub-linear rate for the general convex case. The problems we are facing consists into the optimization of functions that are *nonconvex* and *smooth*, so we don't have the convexity assumptions. In our experiments, we try to apply *SARAH* algorithm to nonconvex and smooth optimization problems (detailed in [5]), observing if the smoothness condition makes possible to obtain good results also without the convexity assumptions.

3.2 SARAH+

One issue of the *SARAH* algorithm is the value assigned to the m parameter, that indicated the number of iteration to do in the inner loop. Trying several values for this parameter, we can note some sensitivity in terms of performance of the *SARAH* algorithm.
The problem can be limited providing this value in an automatic way, finding an adaptive choice: this is done in the *SARAH+* variant of the previous algorithm. In order to do that, we introduce a *stopping criterion* that looks at the square of the norm of v_t

$$||v_{t-1}||^2 < \gamma||v_0||^2 \quad . \quad (9)$$

With the introduction of this modifications, *SARAH+* makes possible to upper bound the size m of the inner

loop, obtaining an earlier termination of the process; it makes unnecessary the careful choice of the parameter m.
What is relevant is the value assigned to the introduced parameter γ. Referring at the experiments shown in [4], we know that *SARAH+* is robust with respect to the choice of γ value.
From the same article [4], we know that $||v_t||^2$ decreases in outer iterations of *SARAH+*, while the same decreases consistently in expectation in the inner loops.

4 SpiderBoost Algorithm

The second algorithm we analyze is *SpiderBoost*, a modified version of the pre-existent *SPIDER*.
SPIDER represents another method for variance reduction which uses the same gradient estimator of *SARAH*, while it adopts a *normalized gradient update*.
In order to be able to reach the convergence, *SPIDER* needs a very restrictive stepsize η that, even if it's possible, makes useless any algorithm's attempt to make consistent progresses.
SpiderBoost is the answer to this issue as detailed in [7] ; it allows to use a much larger stepsize thanks to a new convergence analysis idea: the increments of variables at each entire inner loop iteration are considered, instead of at each single variations.
Starting from the same problem (1), we consider the usual gradient estimator (8). For each iteration k of the inner loop, instead, we consider a sampled index set S and we construct the corresponding estimator as:

$$v_k = \frac{1}{|S|}\sum_{i\in S}[\nabla f_i(x_{k-1}) + v_{k-1}] \quad . \tag{10}$$

This formula allows us to use fresh information and to obtain more accurate estimation for the full gradient. United to the possibility of larger stepsize, *SpiderBoost* algorithm shows considerably progresses per iteration, especially in the initial optimization phase, where the estimated gradient norm $||v_k||$ is large. In the meanwhile, it maintains the same near-optimal oracle complexity.

4.1 SpiderBoost: Convergence Analysis

Now we study the convergence and the complexity of *SpiderBoost* algorithm, adopting the assumptions:

- The objective function is *bounded below*;
- Each gradient ∇fi is *L-Lipschitz continuos*.

Let us solve the problem (1) specifying the value for the parameters $q = |S| = \sqrt{n}$ and $\eta = \frac{1}{2L}$, where q is the dimension of the inner loop, n is the problem's dimensionality and η corresponds to the stepsize used for the updates.
The corresponding output, that we define as x_ξ, satisfies:

$$\mathbb{E}||\nabla f(x_\xi)|| \leq \epsilon \tag{11}$$

provided that the total number of iterations K satisfies:

$$K \geq O(\frac{L(f(x_0) - f^*)}{\epsilon^2}) \quad . \tag{12}$$

The algorithm reaches the following overall *Stochastic First-Order Oracle Complexity* :

$$O(\sqrt{n}\epsilon^{-2} + n) \quad . \tag{13}$$

5 SNVRG Algorithm

The last algorithm considered in this analysis is the *Stochastic Nested Variance Reduction* (*SNVRG*) for nonconvex optimization problems.
Also in this case, the aim is to improve the dependence of the gradient complexity on n and ϵ, following the technique of variance reduction. The gradient complexity, indeed, can be saved in the *history information* of the algorithm as *reference*. While the *SVRG* uses two reference points, the core of *SNVRG* consists into considering $K+1$ *reference points* and $K+1$ *reference gradients*, in order to build a semi-stochastic gradient and making possible a faster decaying of the variance at each iteration.
In case of smooth nonconvex function, the algorithm improves the best known gradient complexity of *SVRG*, in fact it converges to an ϵ -approximate first-order stationary point value that within:

$$\tilde{O}(n \wedge \epsilon^{-2} + \epsilon^{-3} \wedge n^{\frac{1}{2}}\epsilon^{-2}) \quad , \tag{14}$$

where $\tilde{O}$ assumes the use of the logarithmic factors and the $\wedge$ symbol indicates the use of the minimum function (e.g. a$\wedge$b = min(a,b)).
So this method observes improvements in optimization of the entire problem (1), without making additional assumptions beyond smoothness and bounded stochastic gradient variance.

The key component of the main algorithm is called *One-epoch-SNVRG* where we can see the use of the $K+1$ reference points and the $K+1$ reference gradients. In order to better understand the updating rule, we need to define some parameters: K is the number of nested loops of the algorithm, $l \in [K]$, T_l are loop parameters, B_l are batch parameters and B represents the base batch size.
For each $t = 0, ..., \prod_{l=1}^{K} T_l - 1$, we have that the updated value x_t has $K+1$ reference points, defined as $\{x_t^{(l)}\}$. Starting from them, we compute the $K+1$ reference gradients, $\{g_t^{(l)}\}$, for each x_t. In particular the reference gradients are computed as:

$$g_t^{(0)} = \frac{1}{B} \sum_{i \in I} \nabla f_i(x_0) \tag{15}$$

$$g_t^{(l)} = \frac{1}{B_l} \sum_{i \in I_l} \nabla f_i(x_t^{(l)}) - \nabla f_i(x_t^{(l-1)}) \tag{16}$$

where I and I_l are random index sets with cardinality B and B_l respectively. Composing the two parts we can obtain:

$$v_t = \sum_{l=0}^{K} g_t^{(l)} \quad . \tag{17}$$

Finally we have the update rule, that consists in:

$$x_{t+1} = x_t - \eta v_t \quad . \tag{18}$$

To perform the complete *SNVRG* algorithm, we consider *One-epoch-SNVRG* as *building block*: it's necessary to execute it for each iteration.

5.1 SNVRG: Convergence Analysis

In order to analyze the convergence of the entire *SNVRG* algorithm, we first focus on *One-epoch-SNVRG*, setting the parameters values as shown in [2].
We assume that F has *averaged L-Lipschitz gradient*, $B \geq 2$, $K = \log\log B$, $M = 6L$, $T_1 = 2$, $B_1 = 6^K B$ and

$$T_l = 2^{2^{l-2}}, \qquad B_l = 6^{K-l+1} \frac{B}{2^{2^{l-1}}}. \tag{19}$$

Assuming also that F has stochastic gradient with bounded variance σ^2 and setting precise values for the remaining parameters, as defined in [2], it's demonstrated that the output y_{out} of the entire *SNVRG* algorithm satisfies what follows:

$$||\nabla F(y_{out})||^2 \leq \epsilon^2 \tag{20}$$

with less than the following stochastic gradient computations:

$$O\left(\log^3 \left(\frac{\sigma^2}{\epsilon^2} \wedge n \right) \left[\frac{\sigma^2}{\epsilon^2} \wedge n + \frac{L \nabla_F}{\epsilon^2} \left(\frac{\sigma^2}{\epsilon^2} \wedge n \right)^{\frac{1}{2}} \right] \right) \quad . \tag{21}$$

Furthermore, considering σ^2, L, ∇_F constants and assuming $\epsilon \ll 1$, then the formula (21) can be simplified to

$$\tilde{O}(\epsilon^{-3} \wedge n^{\frac{1}{2}} \epsilon^{-2}) \quad . \tag{22}$$

6 Experiments

To support the theoretical analyses, in this subsection, we present our empirical experiments, comparing the performance of the presented algorithms and other baseline algorithms for solving the *logistic regression with nonconvex regularizer* (3) problem and the *nonconvex robust linear regression* (5) problem, using three different datasets.
We consider different stepsizes (reported in the table below, where LLR: logistic regression regularized, and RLG: robust linear regression), with the aim to reach the best result as possible for all the different methods employed. For the problem that uses (5), we initialize all the algorithms to a zero vector, in order to give to all the methods the same starting point. We act different for the problem in which (3) is involved: here in fact the solution is very close to the origin. For this reason, in order to observe the methods' performances we made the algorithms start from a

initial point slightly different from zero.
Next, we set a fixed mini-batch size 256 and we make sure that all the algorithms pass over the entire datasets twice in each epoch.

	dataset 1		dataset 2		dataset 3	
	LLR	RLG	LLR	RLG	LLR	RLG
SGD	0.0001	0.001	0.001	0.001	0.001	0.002
SAG	0.0001	0.001	0.001	0.001	0.001	0.0001
SAGA	0.0002	0.001	0.001	0.01	0.01	0.01
SVRG	0.0005	0.01	0.00005	0.001	0.01	0.005
SARAH	0.001	0.01	0.0001	0.001	0.01	0.01
SARAH+	0.001	0.01	0.0001	0.001	0.05	0.01
SpiderB	0.001	0.01	0.001	0.01	0.02	0.005
SNVRG	10	5	50	10	3	10

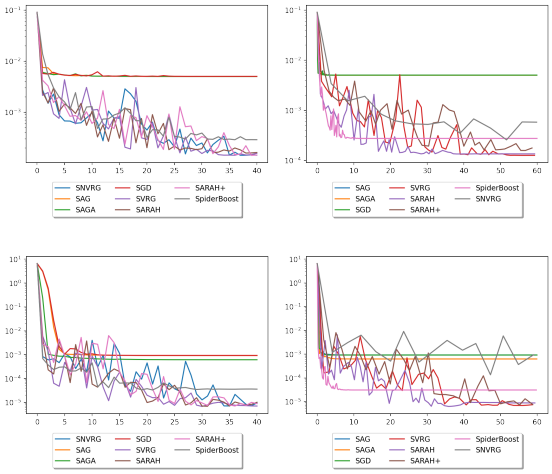
6.1 Datasets

The practical performances of the different algorithms are tested on three datasets that are composed of images divided in two classes.
The first one [1] is composed of 12500 images of cats and 12500 images of dogs; for computational reasons we select the first 7000 elements of both classes. Each image is, then, converted from RGB to Greyscale and resized to a 100x100 pixel format. After flattening each example to an array composed of 10000 features, PCA is applied in order to select only the first 300 principal components.
The second dataset [6] is composed of 256000 images of waste, divided in biodegradable material and non-biodegradable material classes. The third dataset [3] consist of 10000 images of fish as opposed to 10000 random images. As for the first one, from both are selected 7000 images per classes, the images processing phase for those is the same done for the first dataset, except that PCA select the first 1000 principal components in the third dataset.

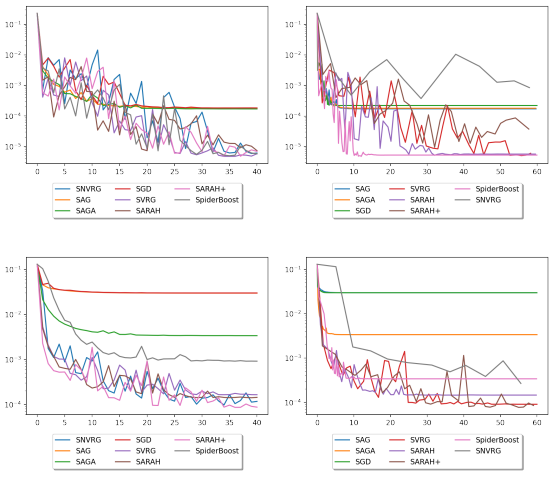
6.2 Results

The analyzed algorithms are applied to solve the two smooth non-convex problems, each with all the three datasets. In the figures below, the experiment results obtained with the *logistic regression with non-convex regularizer* (4) are reported on top, and the ones obtained with the *nonconvex robust linear regression* (5) are on the bottom. The y-axis semi-logarithmic plot of the distance between the loss function values and the real minimum, plotted two times per epoch, is reported on the left column. The same logarithmic residues are plotted on the right figures, but now compared to the computational time calculated two times per epoch (seconds). In these plots are shown only the first 60 seconds of the entire running time.

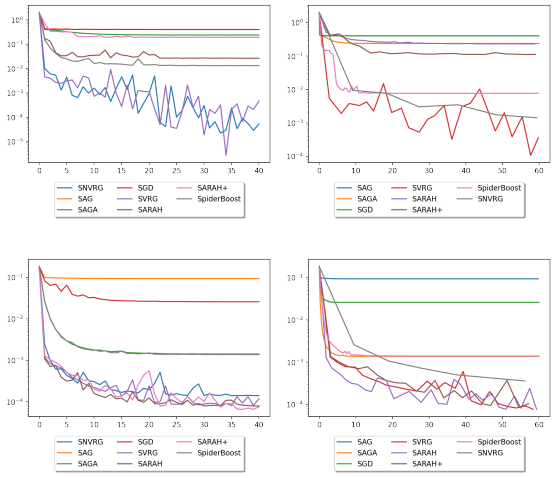
Dataset: Cats vs Dogs



Dataset: Non and Biodegradable Material



Dataset: Fish vs No Fish



The left figures confirm the theoretical results on the gradient complexity of the different algorithms, for non-convex and smooth functions, to achieve an ϵ-approximate stationary point. Below are reported the theoretical complexities for non-convex cases.

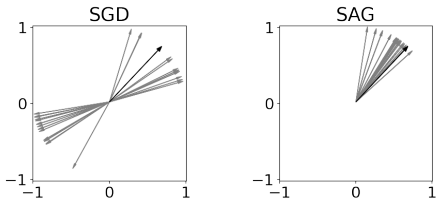
Algorithm	Gradient Complexity (non-convex smooth function)
SGD	$\mathcal{O}(1/\epsilon^4)$
SAG	N/A
SAGA	$\mathcal{O}(n + n^{2/3}/\epsilon)$
SVRG	$\mathcal{O}(n + n^{2/3}/\epsilon^2)$
SARAH	$\mathcal{O}(n^{1/2}/\epsilon \vee n)$
SARAH+	N/A
SpiderBoost	$\mathcal{O}(n + n^{1/2}/\epsilon^2)$
SNVRG	$\mathcal{O}((n + n^{1/2}/\epsilon^2)\log(n))$

Indeed, the gradient complexities of *SGD*, *SAG*, *SAGA* are smaller than the ones of the other algorithms analyzed. The consequence is that, at certain moment, the gradient becomes almost zero and all the algorithm's subsequent steps are almost irrelevant.
The plots of algorithms' performances in the solving of the *logistic regression* problem for the third dataset, show that the curve representing *SAGA* and *SpiderBoost* seem to follow the same direction. For the remaining two datasets, they describe two different paths where *SAGA* stays over *SpiderBoost*.
The algorithms developed for reducing the variance of the steps follow similar walks and, in general, *SVRG* and *SNVRG* reach lower values than the others.
Focusing now on the *robust linear regression* problem, the conclusions are basically the same, what changes is that *SARAH* is the algorithm that reaches lower values.
The figures show, also, the difference between the methods which use a full gradient estimation over the whole minibatch and the other ones. In fact, the former are much more able to find the solution of the minibatch. Each time, indeed, the minimum value for the loss function in a determined minibatch is reached, then the process is restarted with another one. The reduction of the loss function's fluctuation, that is easily visible in the figures, associated with the use of minibatches, is given by the iterative diminishing of the learning rate, that allows us to make corrections with respect to each used minibatch. In this way we are able to reach a good approximation for the final value of the loss function.
However, the time graph on the right side does not show the same good results for SNVRG that is, in fact, much slower than other algorithms, which reach better values at the same time.
The thing that is common in all the figures is that *SGD*, *SAG* and *SAGA* turn out to be quite slow in terms of computational time, compared to the other algorithms.
Indeed, their constant trend suggests that their progress is minimal as time goes on, fact that confirms the theoretical results on the gradient complexities said before.
Focusing on the *logistic regression with non-convex regularizer* problem, *SNVRG* is quite slow in achieving its just discussed good results: in 60 seconds it is able to reach similar values, if not worse, than the above-mentioned algorithms.
This fact could be expected: its implementation is computationally expensive for the higher number of loops, compared to the algorithms analyzed.
In the *robust linear regression* problem, *SGD* and *SAG*, for the same reasons as before, are still the slowest in terms of algorithms' progresses across all three datasets. Those which reach a shorter distance from the minimum point after 60 seconds are *SARAH*,

SARAH+ and *SVRG*.

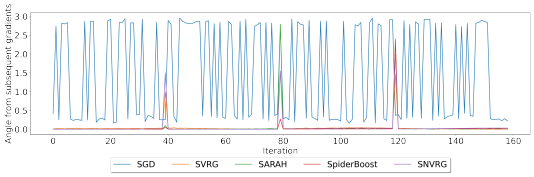
6.3 Variance reduction

All the algorithms, analyzed up to now, aim to reduce the variance introduced by the stochastic part of the algorithms.
This variance can be interpreted as *"how likely we will take a step in the right direction"*: the less the variance, the likely is the step to be in the right direction.
This can be show already with the most basic algorithm, SAG, in the following images:

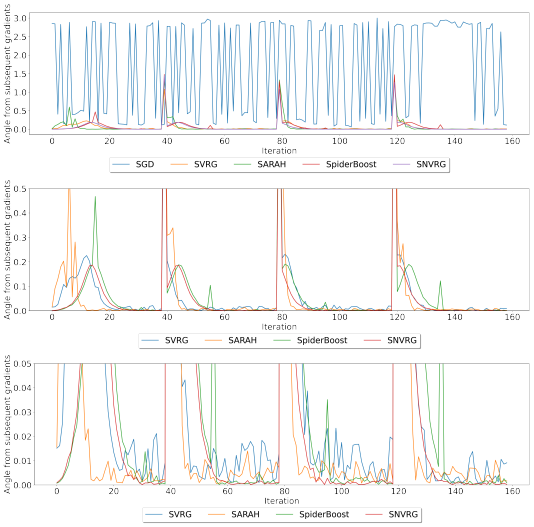


The arrows represent a sample of 30 possible directions (gradients), using the *"Cat vs Dog"* dataset reduced to 2 dimensions, thanks to the application of PCA, using the robust linear regression loss, starting from the same initial point. Clearly, SAG (which for generate this plot make use only of a subset of the whole gradients which should use) reduces the likelihood to perform a step in a direction which is very far from the correct one.
To further demonstrate this fact, we can consider that, if we have a very high-variance estimator, than 2 following gradients have very different directions, and therefore the angle between them increases.
Following plots represent the iterations over the x-axis, and the angle between two consecutive gradients (in radiants) on the y-axis.

"Cat vs Dog" using robust linear regression

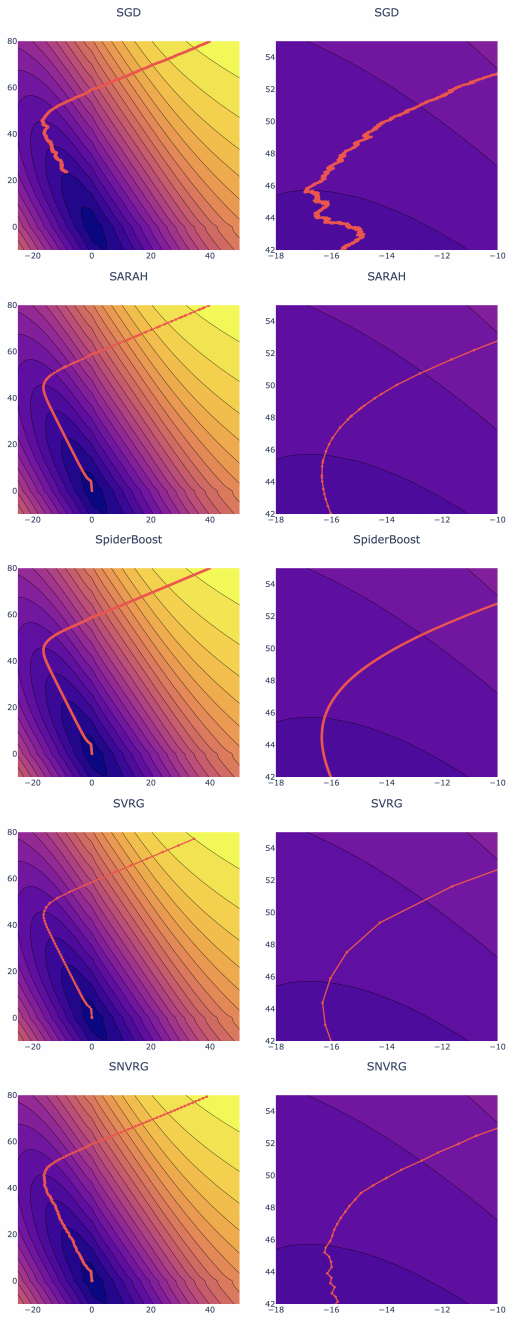


"Fish or not" using cross entropy

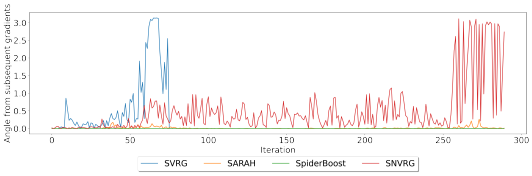


The blue line corresponds to the SGD algorithm: clearly, this algorithm is likely to calculate consecutive gradient with either $\sim 11°$ or $\sim 160°$ angle between them, and this trend increases the closer we are to the minimum. On the other hand, about all the others algorithms, shown in the plots, there is clear evidence of variance reduction: in fact all the updates have close to $\sim 1°$ angle between them, except for some spikes, which comes when we fed the algorithms a new mini-batch. The current minimum might be far from the minimum of the previous one but, however, most of those spikes are around $\sim 15°/20°$ and, therefore, not too different from the previous one.
To finally demonstrate this fact, following are contour plots of the dataset *"Cats vs Dogs"*, reduced to 2

dimensions with PCA method, using the robust linear regression loss.

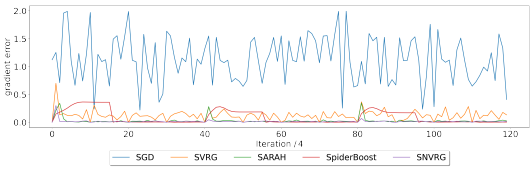


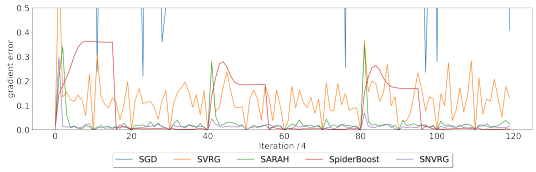
The images on the left represent the trajectory chosen by each algorithms, where instead the images on right are enlargements of the trajectory on the right, which show how SGD has very noisy gradient estimation (high variance), compared to all the others. Furthermore, as previously said, SNVRG does not work properly on very low dimensions, in fact also his path is pretty noisy, and when it reaches the minimum, it jumps from one side to the other, thing that does not appear to be true in higher dimensions (≥ 10).
This is also proven by the plot of the angle between gradients for this run:



7 Gradient estimation

Another aspect that is influenced by the reduction of the variance is the error of the estimated gradient with respect to the correct gradient.
In fact, if an estimator has high variance, than the error between the estimated gradient and the true gradient is high (high variance); instead, a good estimator has a smaller error, and therefore estimates the gradient with better precision (small variance).
To demonstrate this fact, at each step of the analyzed algorithm, we keep track of the estimated gradients and of the current point on which those gradients are been calculated. Then, a posteriori, we recalculate the correct gradient, in order to find the distance between them (to avoid problems with unbiased estimators we scaled the stored estimated gradient and the newly calculated gradient to a unit vector).
Following is the plot of the first 480 iterations of the algorithm on the "Cat vs Dog" dataset:

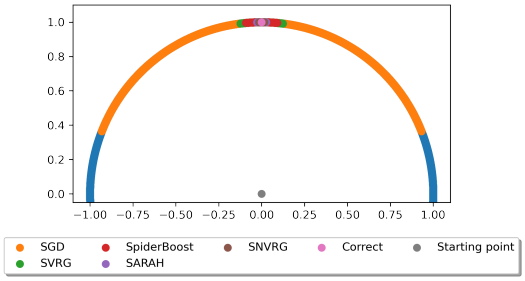




The mean of the errors are reported below:

Algorithm	MSE Gradient Est.
SGD	1.129
SVRG	0.125
SARAH	0.027
SpiderBoost	0.092
SNVRG	0.015

The results clearly show that *SNVRG* almost outperforms all the others, and the outcome is even clearer from the next plot: each color represent an algorithm and its variance (in other words, where, with that variance, you might go around the correct direction)



Also this estimation is been calculated using the first 480 iterations on the "Cat vs Dog" dataset, which are the safest since very far from the minimum. For this reason, even an estimator with large variance might have very good guesses and, therefore, the obtained result has to be considered a lower bound (for that dataset, since this might change from one to another).

8 Final Conclusions

In this paper we proposed some stochastic variance reduced gradient methods for smooth non-convex optimization problems. All the analyzed algorithms represent, in many cases, valid alternatives to improve the performance of *Stochastic Gradient Descent*, by reducing its variance and therefore improving its convergence.

These considerations are observed also in the empirical experiments part of this paper. We have observed that all these algorithms in the non-convex case perform well, even SARAH which is originally used for convex functions problems. This result can be given from the particular structure of the loss function we are using to face the optimization problems: their shapes, in fact, show some convex areas for which *SARAH* is optimized. Another aspect that we can note from our analysis is the better functioning of *SNVRG* in presence of a large number of dimensions. However, we must keep in mind that *SNVRG* requires the use of many hyperparameters. We can instead say that *SpiderBoost* also works quite well, but that due to its sensitivity in the choice of S and Q and its performance almost on a par with *SARAH*, it cannot be considered a very good candidate.

References

[1] *Cats vs Dogs dataset*. URL: https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset?resource=download.

[2] Dongruo Zhou, Pan Xu, Quanquan Gu. «Stochastic Nested Variance Reduction for Nonconvex Optimization». In: (2018).

[3] *Fish vs No Fish dataset*. URL: https://www.kaggle.com/datasets/giannisgeorgiou/fish-or-no-fish-simple-images.

[4] Lam M. Nguyen, Jie Liu, Katya Scheinberg, Martin Takáč. «SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient». In: (2017).

[5] Lam M. Nguyen, Marten van Dijk, Dzung T. Phan, Phuong Ha Nguyen, Tsui-Wei Weng, Jayant R. Kalagnanam. «Finite-Sum Smooth Optimization with SARAH». In: (2019).

[6] *Non and Biodegradable Material dataset*. URL: https://www.kaggle.com/datasets/rayhanzamzamy/non-and-biodegradable-waste-dataset.

[7] Zhe Wang, Kaiyi Ji, Yi Zhou, Yingbin Liang, Vahid Tarokh. «SpiderBoost and Momentum: Faster Stochastic Variance Reduction Algorithms». In: (2019).