Stochastic Zeroth-Order Optimisation Algorithms with Variance Reduction

Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne

Thése de Master de

Ahmad Ajalloeian

Supervisée par

Prof. Martin Jaggi, Machine Learning and Optimization, EPFL
Dr. Sebastian U. Stich, Machine Learning and Optimization, EPFL

Lausanne, EPFL, 2019

Abstract

Introduction of optimisation problems in which the objective function is black box or obtaining the gradient is infeasible, has recently raised interest in zeroth-order optimisation methods. As an example finding adversarial examples for Deep Learning models (Chen et al. (2017); Moosavi-Dezfooli et al. (2016)) is one of the most common applications in which zeroth-order methods could be used. These optimisation methods use only function values at certain points to estimate the gradient. Most current approaches iteratively sample a random search direction along which they compute an estimation of the gradient (Nesterov and Spokoiny (2017); Conn et al. (2009); Wibisono et al. (2012)). However, due to the high variance in the search direction, these methods usually need d times more iterations than the standard gradient methods, where d is the dimensionality of the problem. So it seems that the main effort for improving the zeroth-order methods should be in reducing the variance of the gradient estimate.

In this work we will analyse the gradient-free oracle which uses random directions sampled form a Gaussian distribution. Our analysis shows that in smooth and strongly convex setting, we have a convergence rate of $\mathcal{O}\left(\frac{d}{T}\right)$ which clearly shows the dependency to the dimension of the problem. Furthermore we propose some variance reduction methods to make the zeroth-order optimisation faster. We experiment our proposed methods in Python to compare their convergence in stochastic and non-stochastic setting. Our empirical results show that in a setting that number of allowed function evaluation is fixed, using a variance reduction method (e.g. momentum) can make the convergence of zeroth-order methods happen faster.

Keywords: Stochastic Gradient Descent, Machine Learning, zeroth-order optimisation, convex function, variance reduction methods

Contents

Acknowledgment

I would like to thank Martin and Sebastian for their kindness and support during all stages of my Master Thesis. They gave me a great opportunity to discover a new research field. It was a pleasure for me to work with Sebastian and learn from his experience and I appreciate his kindness and patience during long discussions we had for my Master Thesis. I would like to also thank Thijs Vogels for his interesting ideas and suggestions.

I want to explain my gratitude to all members of the *Machine Learning and Optimisation laboratory* with whom I had a great time during the past semester. Their passion for work and their openness for discussions was always a motivation for me to come to the lab and share my moments with them. I also want to thank my friends with whom I had a lot of fun moments during my studies.

Finally, I would like to thank my lovely parents who always supported me during my Master studies. I am always thankful to them for all the inspiration and motivation they gave me during my studies and the lessons they taught me during my life.

Introduction

In the recent years, Machine Learning and Deep Learning were very successful in solving hard problems in computer vision (Krizhevsky et al. (2012)), natural language processing (Vaswani et al. (2017)) and other domains. One of the main reasons of this success was the introduction of efficient optimisation algorithms that could optimise different loss functions being used in Machine Learning. Particularly for most of the loss functions used in Machine Learning, first-order gradient information is easily accessible since the closed form of the gradient of the function can be calculated easily. Therefore having the first-order gradient information, we can use an efficient iterative algorithm like Stochastic Gradient Descent (SGD) to optimise the function. Nevertheless, the introduction of problems where the objective function to optimise is a black box (e.g. Moosavi-Dezfooli et al. (2016); Chen et al. (2017)) has recently raised interest in gradient-free or zeroth-order optimisation algorithms. These algorithms approximate the full gradient via gradient estimators which are only based on the computation of function values (Nesterov and Spokoiny (2017), Ghadimi and Lan (2013); Brent (2002)).

Although many zeroth-order algorithms have been developed and analyzed, they often suffer from a high variance of the gradient estimate which in turn leads to a slow convergence. It seems that these methods need at most d times more iterations to reach convergence compared to standard gradient methods (Nesterov and Spokoiny (2017)). In this thesis, we tackle the problem of high variance of zeroth-order gradient estimators. We draw motivations from variance reduction ideas in the first-order regime and propose schemes which can make zeroth-order algorithms faster with less computation of function values. Instead of forgetting the gradient estimates calculated in the past iterations, we keep them in memory and add them (with some weight) to the current gradient estimate. We also provide a new analysis for the stochastic zeroth-order algorithm (`ZO_SGD`) for smooth and strongly convex functions. Our analysis proves a $\mathcal{O}\big(\frac{d}{T}\big)$ convergence rate for `ZO_SGD`. This further confirms that zeroth-order algorithms have the same convergence rate as standard gradient methods, except for an extra factor which is in the order of the dimensionality of the problem.

This Master Thesis is organised as follows: Chapter 1 gives an introduction to optimisation and Machine Learning. Reader should refer to Section 1.2 and Appendix A for definition and Lemma which were later used in the proofs. Chapter 2 introduces a class of gradient-free oracle that we used in this work and presents several properties for such oracle. In particular this chapter discusses variance upper bound for the gradient-free Oracle. Section 2.4 of this chapter contains our main analysis for `ZO_SGD`. Chapter 3 presents variance reduction schemes for zeroth-order algorithms. Finally, we present our python implementation and experiments in Chapter 4. Our results shows the effectiveness of variance reduction schemes in the convergence of the zeroth-order algorithms.

Chapter 1

Optimization and Machine Learning

Many real world problems can be seen as an optimisation problem which aims to minimize a loss function or maximise a reward function. In this chapter we will give a brief introduction to convex optimisation and how it is used in machine learning. We will also introduce the class of steepest descent algorithms which are the main tool in machine learning applications. Section 1.2 contains the main definitions which will be later used in Chapters 2 and C. In Section 1.3 we discuss briefly about convergence rate of stochastic gradient descent. In Section 1.4.1 we introduce logistic regression, a well known machine learning algorithm that was used in our experiments (Chapter 4)

1.1 Convex optimisation

We consider the problem of finite sum minimisation: given n functions $f_i : \mathbb{R}^d \to \mathbb{R}$ for $i \in [n]$, we want to find the point x^* that minimizes the average. Many problems in Machine Learning can be formulated as above. Formally:

$$x^* = \min_{x \in \mathbb{R}^d} f(x), \qquad f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{1.1}$$

In the convex setting if the derivative of function $f(x)$ is defined over all of its domain $\mathbb{R}^d$ and can be inverted, then the solution x^* is defined by the first-order constraint $\nabla f(x) = 0$. However in many Machine Learning applications like Logistic Regression, SVM, etc there is no closed form solution for x^* from the first-order constraint and therefore we need an alternative algorithm to find the minimizer.

A common algorithm to minimize a function is the steepest descent method. This is an iterative method that in each iteration takes a small step towards the opposite direction of the gradient computed at the current point x_t. Formally we call this the *update* step:

$$x_{t+1} = x_t - \eta \nabla f(x_t) \tag{1.2}$$

where η is the step size. Given the fact that gradient is only a local indication of the direction with the steepest slope, the step size should not be too big. Otherwise we may go to a point with a higher function value and we may never reach the neighborhood of x^*. The steepest descent algorithm with the update step (1.2) is called Gradient Descent. This algorithm is described in Algorithm 1. Under strong convexity (definition 3) and smoothness (definition 2) we are guaranteed to converge after $T = \mathcal{O}(\log \frac{1}{\epsilon})$ iterations to a point x_T such that $f(x_T) - f(x_0) < \epsilon$ (Boyd and Vandenberghe (2004)).

Algorithm 1 Gradient-Descent (GD)

step size η
initialize x_0 randomly
for t *in* $0, 1, ..., T-1$ **do**
  $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
end
return x_T

Despite the fast convergence rate, the update step in gradient descent algorithm is very costly. In each iteration we have to compute the gradient of function f which is equivalent to computing the gradient of all n functions f_i. This could be very costly in Machine Learning applications where n usually represents the number of data points which can be very large. In order to overcome this issue, we can use the Stochastic Gradient Descent algorithm (Algorithm 2) which is the standard practice in Machine Learning and Deep Learning. In each step t, we sample a function f_i uniformly at random and update the iterate as follows:

$$x_{t+1} = x_t - \eta_t \nabla f_i(x) \tag{1.3}$$

In each iteration we compute the gradient of one loss function f_i (instead of n) and therefore our update step will become n times faster. This reduced computational cost is coming in the cost of a new source of noise which is the choice of the random function f_i. However, although the ∇f_i can lead us to a "wrong" direction, it is an unbiased estimator of ∇f i.e. $\mathbb{E}_i \nabla f_i(x) = f(x)$. This means that on average the opposite direction of ∇f_i is giving us the steepest descent direction. To compensate for this extra noise, we usually use a time variant step size for SGD. That is, in the initial steps when we are far from optimum taking large steps is not dangerous and we can make a lot of progress. But as we get close to the neighbourhood of x^* we want to make smaller steps. SGD is a very efficient algorithm and under strong convexity and smoothness it needs $\mathcal{O}(\frac{1}{\epsilon})$ iterations to reach x^* with accuracy ϵ. Here accuracy is the difference between the function value computed at the weighted average of the iterates function value at the initial point (Lacoste-Julien et al. (2012)). Precisely, $f(\bar{x_T}) - f(x_0) < \epsilon$ where $\bar{x_T} = \frac{2}{T(T+1)} \sum_{t=1}^{T} t x_t$. In practice SGD is also used to optimise non-convex functions (e.g. in Deep Learning)

Algorithm 2 Stochastic-Gradient-Descent (SGD)

step size η_t
initialize x_0 randomly
for t *in* $0, 1, ..., T-1$ **do**
  pick index i randomly
  $x_{t+1} \leftarrow x_t - \eta_t \nabla f_i(x_t)$
end
return $\bar{x_T}$

1.2 Notations and definitions

In this section we are going to introduce some of the notations and definitions that are standard in the context of convex optimisation

Notation 1. *Given a function $f : \mathbb{R}^d \to \mathbb{R}$, we denote its minimizer (if existing and unique) by x^* and the minimum function value by f^* i.e. $f^* = f(x^*)$*

Definition 1. *(convexity) We say that a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex iff*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle \tag{1.4}$$

Definition 2. *(L-smoothness) Function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth for some $L > 0$, if for all $x, y \in \mathbb{R}^d$ we have*

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|^2 \tag{1.5}$$

or equivalently

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2 \tag{1.6}$$

Definition 3. *(α-strong convexity) Function $f : \mathbb{R}^d \to \mathbb{R}$ is α-strongly convex for $\alpha > 0$, if for all $x, y \in \mathbb{R}^d$*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2 \tag{1.7}$$

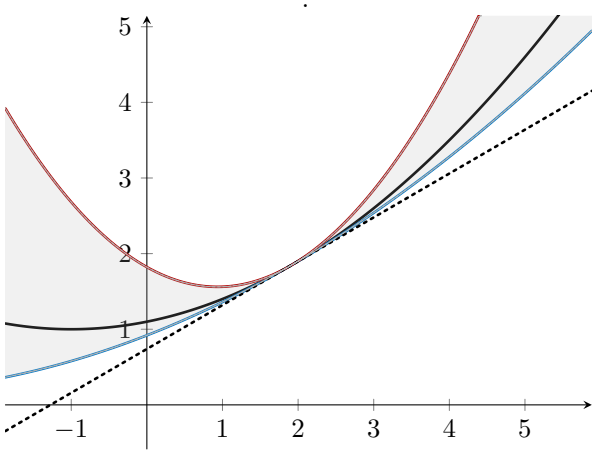


Figure 1.1: Illustration of strong convexity and smoothness. The *black* curve shows the function f and the *dotted black line* shows the gradient tangent line at point $x = 2$. The *red* curve shows the smoothness upper bound and the *blue* curve shows the strong convexity lower bound [1]

Corollary 1. *If f is strongly convex then it is strictly convex. Suppose x^* is a local minimum of a strongly convex function f, for all $x \neq x^*$ we have $\|x - x^*\|^2 > 0$ and $\nabla f(x^*) = 0$. Therefore we have $f(x) > f(x^*)$ and x^* is a global minimum.*

A smooth and strongly convex function is locally sandwiched between two quadratic functions at any point. Smoothness says that at any point x the function f is below a *not-too-steep* tangential paraboloid at $(x, f(x))$ while Strong convexity means that the graph of f is above a *not-too-flat* tangential paraboloid at $(x, f(x))$ (maybe a citation to Opt for ML course lecture notes). This is illustrated in Figure 1.1

1.3 Convergence of SGD

Lacoste-Julien et al. (2012) provides a simple analysis of SGD to derive the $\mathcal{O}\left(\frac{1}{T}\right)$ convergence rate for smooth and strongly convex functions. They use a weighted average scheme which gives a weight of $t+1$ to each iterate x_t at time t. They show that they can obtain the rate of $\mathcal{O}\left(\frac{1}{T}\right)$ with both an easy proof and an easy implementation. We present their main result in the Following theorem:

Theorem 2. *Let f be the objective function defined in (1.1), where each f_i is L-smooth and f is α-strongly convex. Moreover assume that $\mathbb{E}_i \nabla f_i(x) \leq B^2$. If we choose step size in Algorithm 2 to be $\eta_t = \frac{2}{\alpha(t+1)}$ we will have:*

$$\mathbb{E} f(\bar{x_T}) - f(x^*) \leq \frac{2B^2}{\alpha(T+1)} \tag{1.8}$$

where $\bar{x_T} = \frac{2}{(T+1)(T+2)} \sum_{t=0}^{T} (t+1) x_t$

The proof of the above theorem is presented in Section 3 of Lacoste-Julien et al. (2012). The inequality (1.8) not only shows the $\mathcal{O}\left(\frac{1}{T}\right)$ convergence for SGD, but also shows that the convergence rate of SGD is only dependent on the upper bound of the variance of the stochastic oracle (note that $\mathbb{E}\|X - \mathbb{E}X\|^2 \leq \mathbb{E}\|X\|^2$). This is nice because we can easily generalize this result to the case when batch size is bigger than one. As the choice of the indices i is independent, the variance stochastic gradients with a batch size of b is upper bounded by $\frac{B^2}{b}$. Therefore theoretically by increasing the batch size we get a linear improvement in the speed of SGD.

[1] Figure from Cordonnier (2018)

1.4 Optimisation in Machine Learning

Many Machine Learning algorithms rely on convex optimisation to learn patterns from the data. They define a parametrized model that associates a loss function to the observed data. The goal is to find the best parameters that minimize this loss function given the observed data. In the next section we will introduce a well known Machine Learning algorithm which is used for classification tasks. We will show that this algorithm expresses a classification problem as minimizing a convex loss function which can be solved by steepest descent algorithms. We will use this algorithm in Chapter 4 for our experiments

1.4.1 Logistic Regression

Consider a binary classification problem where we want to predict the label (0 or 1) of a given input feature. We have a data matrix A and each row a_i of the data matrix represents the feature vector of a data point and the scalar value b_i shows its corresponding target label. During the training we want to learn a weight vector x that can predict the label of an unseen data point given its feature vector. As we are concerned with a binary classification problem, it is desirable that our predictor gives the *probability* of the two class labels. Hence we will use the sigmoid function which maps a scalar to a probability measure in $[0, 1]$.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{1.9}$$

The probability of label b_i given the feature vector a_i according to this model is:

$$\begin{aligned} p(b_i = 1|a_i, x) &= \sigma(a_i^T x) \\ p(b_i = 0|a_i, x) &= 1 - \sigma(a_i^T x) \end{aligned} \tag{1.10}$$

In the next step we quantize the predicted probability to a binary value typically according to wether the probability is smaller or larger than 0.5.

With the assumption that our training set consists of i.i.d samples $(a_i, b_i)_{i=1}^n$, we can derive the probability of $\mathbf{b}$ (vector of all labels) given A and x. We call this value likelihood of the data set:

$$p(\mathbf{b}|A, x) = \prod_{i=1}^{n} p(b_i|a_i, x) = \prod_{i=1}^{n} \sigma(a_i^T x)^{b_i} \left[1 - \sigma(a_i^T x)\right]^{1-b_i} \tag{1.11}$$

And we consider the negative log likelihood as loss function:

$$\mathcal{L}(x) = \frac{-1}{n} \sum_{i=1}^{n} b_i \ln\left[\sigma\left(a_i^T x\right)\right] + (1 - b_i) \ln\left[1 - \sigma\left(a_i^T x\right)\right] \tag{1.12}$$

We want to maximise the likelihood of the observed data give our model x. This is equivalent to minimizing the negative log likelihood in (1.12). The function $\ln\left[\sigma\left(a_i^T x\right)\right]$ is a combination of a convex and linear function and hence is convex. Consequently the loss in (1.12) is sum of convex functions and hence is convex and can be optimised using steepest descent algorithms (SGD, GD, ...). Moreover, we usually add a regularisation term to the loss function in (1.12) which makes it a strongly convex function and prevents $\|x\|^2$ to become very large and therefore avoids overfitting.

$$\mathcal{L}(x) = \frac{-1}{n} \sum_{i=1}^{n} b_i \ln\left[\sigma\left(a_i^T x\right)\right] + (1 - b_i) \ln\left[1 - \sigma\left(a_i^T x\right)\right] + \frac{\lambda}{2} \|x\|^2 \tag{1.13}$$

The parameter λ is a tunable hyper-parameter which should be chosen by doing cross-validation on training and validation data-sets. In our experiments we set $\lambda = \frac{1}{n}$ as proposed in Bottou (2010).

Chapter 2

Zeroth-order Optimisation Algorithms

Steepest descent methods like gradient descent and stochastic gradient descent play a major role in many learning tasks. However these methods rely on the assumption that we know the closed form version of the gradient of the function which is not always the case. In fact there exist situations in which obtaining the first-order gradient information is computationally infeasible, expensive, or impossible, while the function values can be easily obtained. As an example in in online auctions and advertisement selections, only function values are revealed as feedbacks for algorithm (Wibisono et al. (2012)). As another example, in black-box adversarial attacks to machine learning models we want to find a perturbation θ to add to the data points that could fool the model. This problem could be formulated as a regularized optimisation problem where we want to optimise the parameter θ. In the setting where we can only acquire input-output correspondences of the model, zeroth-order optimisation methods could be used to find the best perturbation (Chen et al. (2017)). As in this methods we do not have access to first-order gradient information, they are also called gradient-free optimisation methods.

In this section we will introduce the zeroth-order optimisation methods. In particular, we will introduce a class of zeroth-order methods that use two function evaluations to estimate the gradient. In section 2.1 we will review the previous works on zeroth-order optimisation and the different oracles that were used in these works. In section 2.2 we introduce the stochastic zeroth-order oracle that we used throughout this project and its important properties. We will also discuss about the variance of this oracle. In section 2.3 we discuss about the variance of this oracle in the general case where we can pick a batch of random functions and a population of random directions. Finally in section 2.4 we will show a simple analysis for deriving the convergence rate of zeroth-order algorithm with such oracle for smooth and strongly convex functions.

2.1 Related work

Derivative-free optimisation methods were among the first schemes being suggested in the early days of the development of optimisation theory (Nesterov and Spokoiny (2017)). In this early stages the simplicity of the computation of function value compared to the computation of gradient vectors was among the main reasons of popularity of these schemes. However, it was soon realized that the theoretical analysis of gradient-free schemes could be cumbersome. Moreover, it was observed that these methods have a way slower convergence rate compared to usual optimisation methods. On the other hand, by the introduction of computationally efficient techniques for calculating gradients, there was practically no good reasons for using derivative-free schemes. Hence, derivative-free optimisation was abandoned (Nesterov and Spokoiny (2017))

However, in the recent years there was a rise of interest in this topic mainly because of several emerging applications. We mentioned some examples of these applications in the beginning of this chapter. Conn et al. (2009) introduces many more examples of problems in which the usage of derivative-free optimisation is essential.

Therefore there were several works in this field in the recent years among which the work by Nesterov and Spokoiny (2017) is of high importance. In this work authors prove new complexity bounds for optimisation methods which are only based on computation of function values. Such bound were

almost absent in the field prior to this work. They introduce a zeroth-order oracle which uses random search directions sampled from Gaussian distribution. They derive upper bounds on the variance of such oracle and also derive asymptotic convergence rates for strongly convex, smooth convex and non-smooth convex functions in the non-stochastic case. They show that in the convex setting, these methods need at most d times more iterations to converge compared to standard gradient methods, where d is the dimension of the problem. Wibisono et al. (2012) investigated this problem more deeply for non-smooth functions. They consider a derivative-free algorithm for stochastic optimisation that uses two function evaluations for estimating the gradient. They show that such algorithms suffer a factor of at most $\sqrt{d}$ in convergence rate for non-smooth functions (i.e. after T iterations these algorithms can reach accuracy of $\mathcal{O}(\frac{\sqrt{d}}{\sqrt{T}})$). They also find an information-theoretic lower bound on the minimax convergence rate and show that their bounds cannot be improved.

It has been well established that the slower convergence rate of the derivative-free methods is mainly because of the high variance of the gradient estimator in the random search direction. Therefore there has been some works to reduce the variance of the zeroth-order gradient estimator. Liu et al. (2018) did a comprehensive theoritical analysis of variance reduced zeroth-order methods. They proposed a variance reduced zeroth-order algorithm called ZO-SVRG which is basically based on the SVRG algorithm introduced by Johnson and Zhang (2013) except that it uses zeroth-order gradient estimates. They proved for non-convex functions that ZO-SVRG gives a convergence rate of $\mathcal{O}(\frac{d}{T} + \frac{1}{b})$, where b is the batch size. They claim that this rate improves over ZO-SGD rate introduced in Nesterov and Spokoiny (2017) which has rate $\mathcal{O}(\frac{\sqrt{d}}{\sqrt{T}})$ except for the constant term $\frac{1}{b}$. However this term becomes small as we increase the batch size. They also claim that compared to the existing zeroth-order algorithms, ZO-SVRG can strike a balance between iteration complexity and function query complexity. Moreover, they also investigate the oracle which uses a population of random directions and show that it can reduce the number of steps required to converge. their empirical results show that a moderate choice of the population size can significantly speed up the convergence of ZO-SVRG.

Gu et al. (2016) use a similar idea for reducing the variance of the zeroth-order oracle. They introduce an asynchronous doubly stochastic zeroth-order algorithm with variance reduction which they call AsyDSZOVR. Their oracle randomly samples coordinates as search directions instead of sampling random vectors from multivariate Gaussian distribution. Their analysis shows that AsyDSZOVR achieves a convergence rate of $\mathcal{O}(\frac{d}{T})$ for smooth and non-convex functions which is an improvement over the rate $\mathcal{O}(\frac{\sqrt{d}}{\sqrt{T}})$ for asynchronous stochastic zeroth-order algorithm.

In this Thesis we mainly focus on variance reduction methods in the smooth and strongly convex setting. This is the setting which is somehow abandoned in the works that investigated variance reduction for zeroth-order methods. As mentioned above, the previous works only showed that certain variance reduction methods can improve the convergence rate of the zeroth-order methods for non-convex functions. Moreover we tried to investigate the usage of population of random directions more in detail in this thesis and compared different ways for using the population. Besides, we also gave a theoretical analysis for the convergence of the stochastic zeroth-order optimisation for smooth and strongly convex function. We derive an asymptotic convergence rate which could be easily generalised to the setting with arbitrary batch size and population size.

2.2 Stochastic zeroth-order oracle

In this work we used the zeroth-order oracle introduced by Nesterov and Spokoiny (2017). This oracle uses two function evaluations to estimate the gradient. At each point x it samples a random vector from a multivariate Gaussian distribution and add it to the point x to get a second point. Then the estimate is calculated using these two points. Let us define this oracle as follows:

Definition 4. *Let random vector $u \in \mathbb{R}^d$ have Gaussian distribution with mean zero and variance identity i.e. $u \sim \mathcal{N}(0, \boldsymbol{I})$. For $\mu \geq 0$, we can define the following stochastic zeroth-order oracle.*

$$g_{\mu,i}(x, u) = \frac{f_i(x + \mu u) - f_i(x)}{\mu} u \tag{2.1}$$

where i is the index of the random function uniformly sampled from the set $\{1, 2, ..., n\}$

Parameter μ is called the smoothing parameter. It actually determines the norm of the guassian noise which is added to the point x. Therefore if we choose a smaller μ then the two points x and $x+\mu u$ will be close to each other and the gradient estimate will be more accurate (Nesterov and Spokoiny (2017)). In the extreme case, when $\mu \to 0$ then we will get the directional derivative of function f along $u \in \mathbb{R}^d$ that is: $g_{0,i}(x,u) = f_i^{'}(x,u)u$ where $f_i^{'}(x,u) = \langle \nabla f_i(x), u \rangle$ is the directional derivative of f_i along u.

Next, we will define the Gaussian approximation of a function.

Definition 5. *(Gaussian approximation) consider a function $f\colon \mathbb{R}^d \to \mathbb{R}$ which is differentiable along any direction at each point $x \in \mathbb{R}^d$. We can define its Gaussian approximation f_μ as follows*

$$f_\mu(x) = \mathbb{E}_u\big[f(x+\mu u)\big] = \frac{1}{(2\pi)^{\frac{d}{2}}} \int f(x+\mu u) e^{\frac{-1}{2}\|u\|^2} du, \qquad \mu > 0 \tag{2.2}$$

The function f_μ is very important in the context of gradient free optimization. In fact the zeroth-order oracle introduced in Definition 4 is an unbiased estimator of the function f_μ i.e. $\mathbb{E}_{u,i}[g_{\mu,i}(x,u)] = \nabla f_\mu(x)$. In the next section we are going to introduce some of the properties of this function. All of the results are taken from Nesterov and Spokoiny (2017).

2.2.1 Properties of function f_μ

The following lemma says that the smoothed version of a function has always larger values.

Lemma 3. *[Nesterov and Spokoiny (2017)] For $\mu > 0$ the function f_μ is always differentiable and we have:*

$$f_\mu(x) \geq f(x) \tag{2.3}$$

In general f_μ has better properties than f. At least all of the initial properties of f are preserved by f_μ with $\mu \geq 0$. Specifically:

- if f is convex, then f_μ is also convex
- if f is Lipschitz continuous with L_0 then f_μ is also Lipschitz continuous with $L_0(f_\mu)$ and we have: $L_0(f_\mu) \leq L_0$
- if f is L_1-smooth then f_μ is also smooth with parameter $L_1(f_\mu)$ and we have: $L_1(f_\mu) \leq L_1$

It turns out that the gradient of f_μ is Lipschitz-continuous even if the gradient of f is not which means that function f_μ is smooth even when original function is not. We will formalize this in the following Lemma.

Lemma 4. *[Nesterov and Spokoiny (2017)] Let f be Lipschitz with parameter L_0 and $\mu > 0$. Then f_μ is smooth with parameter $L_1(f_\mu)$ and we have:*

$$L_1(f_\mu) = \frac{\sqrt{d}}{\mu} L_0 \tag{2.4}$$

In the next Lemma we will show some useful results relating function f_μ to f.

Lemma 5. *[Nesterov and Spokoiny (2017)] Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be L-smooth, then*

$$|f_\mu(x) - f(x)| \leq \frac{\mu^2}{2} L d, \qquad \forall x \in \mathbb{R}^d \tag{2.5}$$

further we can establish following inequalities between gradient of f_μ and gradient of f:

$$\|\nabla f_\mu(x) - \nabla f(x)\| \leq \frac{\mu}{2} L (d+3)^{3/2} \tag{2.6}$$

$$\|\nabla f(x)\|^2 \leq 2\|\nabla f_\mu(x)\|^2 + \frac{\mu^2}{2} L^2 (d+6)^3 \tag{2.7}$$

One of the consequences of the above lemma is that smaller values for μ gives a more accurate gradient estimate.

2.2.2 Upper bounds on the variance of the gradient free oracle

Nesterov and Spokoiny (2017) establishes several useful upper bounds on the second moment (and hence the variance) of the gradient-free oracle. As we stated earlier, high variance of the gradient-free oracle is the main cause of the slow convergence rate of the zeroth-order methods. Moreover Theorem 2 shows that the convergence rate of SGD is dependent to the variance of the stochastic gradient. Therefore by deriving upper bounds for the variance of the zeroth-order oracle we will be able to follow the same analysis as SGD for the zeroth-order methods. The following Lemma introduces the variance upper bound in two different forms. For simplifying the notation we will drop the subscript μ from now on. So instead of $g_{\mu,i}(x,u)$ we will use $g_i(x,u)$.

Lemma 6. *[Nesterov and Spokoiny (2017)] Let f_i be L-smooth. Then we will have the following:*

$$\mathbb{E}_{u,i}\left\|g_i(x,u)\right\|^2 \leq \frac{\mu^2}{2}L^2(d+6)^3 + 2(d+4)\mathbb{E}_i\left\|\nabla f_i(x)\right\|^2, \qquad \forall x \in \mathbb{R}^d \tag{2.8}$$

We can also write the RHS of the above inequality in terms of the gradient of Gaussian approximation of the function:

$$\mathbb{E}_{u,i}\left\|g_i(x,u)\right\|^2 \leq 3\mu^2L^2(d+4)^3 + 4(d+4)\mathbb{E}_i\left\|\nabla f_{\mu,i}(x)\right\|^2, \qquad \forall x \in \mathbb{R}^d \tag{2.9}$$

2.3 General gradient free oracle

The oracle introduced in Definition 4 samples only one random function f_i and one random direction u. However this could be generalized to the case where the oracle samples arbitrary number of random functions and random directions. As we have two sources of randomness here, namely the choice of index i and random direction u, we have two options for generalising the oracle. The main difference between these two options is that one samples only one random direction for the whole batch while the other samples fresh random directions for each element of the batch. Given a set $\mathcal{I} \subset [n]$ of indices and a set $\mathcal{D}$ of random directions sampled from standard Gaussian[1] we can define these two options:

- **Option I: fixed random direction for the whole batch:** In this Oracle the set $\mathcal{D}$ is fixed. Then for each element $u_j \in \mathcal{D}$ we select a random batch $\mathcal{I}_j \subset [n]$ with a batch size of $|\mathcal{I}|$ i.e. $|\mathcal{I}_j| = |\mathcal{I}|, \forall j$. This oracle is defined as:

$$g_{\mathcal{I}}(x,\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{j=1}^{|\mathcal{D}|}\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}_j} g_i(x,u_j) \tag{2.10}$$

- **Option II: different random directions for each index i:** in this oracle for every index i we sample a new set of directions $\mathcal{D}_i$ independently. We assume that $|\mathcal{D}_i| = |\mathcal{D}|, \forall i$ i.e. the population size is equal to $|\mathcal{D}|$.

$$g_{\mathcal{I}}(x,\mathcal{D}_1,...,\mathcal{D}_{|\mathcal{I}|}) = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\frac{1}{|\mathcal{D}|}\sum_{u\in\mathcal{D}_i} g_i(x,u) \tag{2.11}$$

For finding the variance upper bound, we will start by introducing an assumption on the variance of the choice of random functions f_i. This is a standard assumption which has been used in other works as well (e.g. see).

Assumption 1. *Suppose that we have the following upper bound for the variance of f_i:*

$$\mathbb{E}_i\left\|\nabla f_i(x) - \nabla f(x)\right\|^2 \leq \sigma^2 \tag{2.12}$$

In what follows, we will try to find an upper bound on the variance of the option I and option II gradient free oracles under Assumption 1. Nesterov and Spokoiny (2017) gives upper bounds only on the variance (second moment) of the basic oracle $g_i(x,u)$ which has both batch size and population size equal to one. We introduced these upper bounds in Lemma 6. We are going to derive upper bounds for the variance of the general gradient-free oracles introduced earlier. We will mainly use the results of Lemma 6.

[1] In the rest of this report we may refer to $|\mathcal{I}|$ and $|\mathcal{D}|$ as "batch size" and "population size" respectively.

2.3.1 Variance of the option I oracle $g_{\mathcal{I}}(x, \mathcal{D})$

Note that $g_{\mathcal{I}}(x, \mathcal{D})$ defined in (2.10) is an unbiased estimator of $\nabla f_\mu(x)$. Before investigating the variance upper bound, let's first reformulate the gradient free oracle introduced in (2.10). We define:

$$f_{\mathcal{I}}(x) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} f_i(x) \tag{2.13}$$

So we can write:

$$\begin{aligned} g_{\mathcal{I}}(x, \mathcal{D}) &= \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} g_i(x, u_j) \\ &= \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \frac{f_{\mathcal{I}}(x + \mu u_j) - f_{\mathcal{I}}(x)}{\mu} . u_j \\ &= \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} g_{\mathcal{I}}(x, u_j) \end{aligned} \tag{2.14}$$

Under the Assumption 1 we have: $\mathbb{E}_i \left\| \nabla f_i(x) - \nabla f(x) \right\|^2 \leq \sigma^2$. We first try to find an upper bound for variance of the oracle $g_{\mathcal{I}}(x, u)$, i.e. the oracle with batch size of $|\mathcal{I}|$ and population size of one. Using the variance equality $\mathbb{E} \left\| X - \mathbb{E}X \right\|^2 = \mathbb{E} \left\| X \right\|^2 - \left\| \mathbb{E}X \right\|^2$ we have:

$$\begin{aligned} \mathbb{E}_{\mathcal{I},u} \left\| g_{\mathcal{I}}(x, u) - \nabla f_\mu(x) \right\|^2 &\overset{(2.8)}{\leq} \frac{\mu^2}{2} L^2 (d+6)^3 + 2(d+4) \mathbb{E}_{\mathcal{I}} \left\| \nabla f_{\mathcal{I}}(x) \right\|^2 \\ &= \frac{\mu^2}{2} L^2 (d+6)^3 + 2(d+4) \mathbb{E}_{\mathcal{I}} \left\| \nabla f_{\mathcal{I}}(x) - \nabla f(x) \right\|^2 + 2(d+4) \left\| \nabla f(x) \right\|^2 \\ &\leq \frac{\mu^2}{2} L^2 (d+6)^3 + 2(d+4) \frac{\sigma^2}{|\mathcal{I}|} + 2(d+4) \left\| \nabla f(x) \right\|^2 \end{aligned} \tag{2.15}$$

Since the elements in $\mathcal{D}$ are independent, we have:

$$\begin{aligned} \mathbb{E}_{\mathcal{I},\mathcal{D}} \left\| g_{\mathcal{I}}(x, \mathcal{D}) - \nabla f_\mu(x) \right\|^2 &\leq \frac{\mu^2}{2 |\mathcal{D}|} L^2 (d+6)^3 + 2(d+4) \frac{\sigma^2}{|\mathcal{I}| \, |\mathcal{D}|} + \left(\frac{2(d+4)}{|\mathcal{D}|}\right) \left\| \nabla f(x) \right\|^2 \\ &\leq \mathcal{O}\Big(\frac{\mu^2 L^2 d^3}{|\mathcal{D}|} + \frac{dA}{|\mathcal{D}| \, |\mathcal{I}|} + \frac{d}{|\mathcal{D}|} \left\| \nabla f(x) \right\|^2 \Big) \end{aligned} \tag{2.16}$$

2.3.2 Variance of the option II oracle $g_{\mathcal{I}}(x, \mathcal{D}_1, ..., \mathcal{D}_{|\mathcal{I}|})$

First we reformulate the gradient free oracle $g_{\mathcal{I}}(x, \mathcal{D}_1, ..., \mathcal{D}_{|\mathcal{I}|})$ defined in (2.11). Let $g_i(x, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{u \in \mathcal{D}} g_i(x, u)$. We can write

$$g_{\mathcal{I}}(x, \mathcal{D}_1, ..., \mathcal{D}_{|\mathcal{I}|}) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} g_i(x, \mathcal{D}_i) \tag{2.17}$$

Again note that $g_i(x, \mathcal{D})$ is an unbiased estimator of $\nabla f_\mu(x)$. Since the elements in $\mathcal{D}$ are independent of each other we have:

$$\begin{aligned} \mathbb{E}_{i,\mathcal{D}} \left\| g_i(x, \mathcal{D}) - \nabla f_\mu(x) \right\|^2 &\overset{(2.8)}{\leq} \frac{\mu^2}{2 |\mathcal{D}|} L^2 (d+6)^3 + \frac{2(d+4) \mathbb{E}_i \left\| \nabla f_i(x) \right\|^2}{|\mathcal{D}|} \\ &= \frac{\mu^2}{2 |\mathcal{D}|} L^2 (d+6)^3 + \frac{2(d+4) \mathbb{E}_i \left\| \nabla f_i(x) - \nabla f(x) \right\|^2}{|\mathcal{D}|} + \frac{2(d+4) \left\| \nabla f(x) \right\|^2}{|\mathcal{D}|} \\ &\leq \frac{\mu^2}{2 |\mathcal{D}|} L^2 (d+6)^3 + \frac{2(d+4) \sigma^2}{|\mathcal{D}|} + \frac{2(d+4) \left\| \nabla f(x) \right\|^2}{|\mathcal{D}|} \end{aligned} \tag{2.18}$$

Again note that elements of the set $\mathcal{I}$ are independent and $g_{\mathcal{I}}(x, \mathcal{D}_1, ..., \mathcal{D}_{|\mathcal{I}|})$ is an unbiased estimator of $\nabla f_\mu(x)$. Given that $|\mathcal{D}_i| = |\mathcal{D}|$ for all i we have:

$$\begin{aligned}\mathbb{E}\left\|g_{\mathcal{I}}(x, \mathcal{D}_1, ..., \mathcal{D}_{|\mathcal{I}|}) - \nabla f_\mu(x)\right\|^2 &\leq \frac{\mu^2}{2\left|\mathcal{D}\right|\left|\mathcal{I}\right|}L^2(d+6)^3 + \frac{2(d+4)\sigma^2}{\left|\mathcal{D}\right|\left|\mathcal{I}\right|} + \frac{2(d+4)\left\|\nabla f(x)\right\|^2}{\left|\mathcal{D}\right|\left|\mathcal{I}\right|} \\ &\leq \mathcal{O}\Big(\frac{\mu^2 L^2 d^3}{\left|\mathcal{D}\right|\left|\mathcal{I}\right|} + \frac{d\sigma^2}{\left|\mathcal{D}\right|\left|\mathcal{I}\right|} + \frac{d}{\left|\mathcal{D}\right|\left|\mathcal{I}\right|}\left\|\nabla f(x)\right\|^2\Big)\end{aligned} \tag{2.19}$$

2.3.3 Discussion of the variance bounds

The upper bound on the variance of option I and II gradient free oracle can be written in the following general form:

$$F_1(\mu, L, d, \left|\mathcal{I}\right|, \left|\mathcal{D}\right|) + F_2(d, \left|\mathcal{I}\right|, \left|\mathcal{D}\right|)\sigma^2 + F_3(d, \left|\mathcal{I}\right|, \left|\mathcal{D}\right|)\left\|\nabla f(x)\right\|^2 \tag{2.20}$$

Each of the three terms in (2.20) seems to be unavoidable:

- Consider as an example the simple case in which we have a population size of one and function $f = f_1$ i.e. we don't have any randomness in i. Therefore the variance due to the choice of i will be zero and hence: $\sigma^2 = 0$. Hence the second term in (2.20) will be equal to zero. However in this case we still have the randomness in u (the choice of random direction to pick), so the third term in (2.20) is necessary to upper bound the variance of the gradient free oracle. In the extreme case where $\mu = 0$ this simple gradient free oracle is equivalent to the directional derivative of function f along u, i.e. $g_0(x, u) = \langle \nabla f(x), u\rangle\, u$. In such a case the variance of the oracle is only dependent on d and $\left\|\nabla f(x)\right\|^2$. This is also illustrated in inequality (32) of Nesterov and Spokoiny (2017). We have:

$$\mathbb{E}_u g_0(x, u) \leq (d+4)\left\|\nabla f(x)\right\|^2 \tag{2.21}$$

- To show the essence of first two terms in the bound (2.20) it is enough to consider oracle $g_i(x, u)$, which has batch size and population size equal to one, at point x^* where $\nabla f(x^*) = 0$. At this point the third term in (2.20) will be equal to zero. The second term shows the increase of variance due to the choice of f_i (the term A) and random direction (the term d) while the first term shows the increase of the variance due to the Gaussian smoothing ($\mu > 0$).

Also comparing option I oracle defined in (2.10) with option II defined in (2.11), we can see that in the oracle option I batch size only helps to reduce the second term in (2.20) whereas it reduces all three terms in the option II oracle. Hence oracle option II is more effective for reducing the variance. Intuitively this conclusion makes sense as well, by sampling fresh random directions for each element of the batch we will get more information about the true gradient of f at point x.

2.4 Proof for Strongly convex case under bounded Variance assumption

In what follows we will focus on the analysis of the stochastic zeroth-order optimisation with batch size and population size of one. We will derive the convergence rate for smooth and strongly convex functions. Our main assumption is bounded variance assumption introduced in Assumption 1. Theorem presents our main result:

Theorem 7. *Suppose that we have the objective function defined in* (1.1) *and each f_i is L-smooth and f is α-strongly convex. Further suppose that for optimising f we will use SGD introduced in Algorithm 2 except that we use the stochastic zeroth-order oracle in* (2.1) *for our updates. By choosing $\mu \leq (Ld^{1.5})^{-1}$ and $\eta_t = \frac{2}{\alpha(c+t)}$, where $c = 32(d+4)\kappa$ and $\kappa = \frac{L}{\alpha}$, after T iterations of the algorithm we will have:*

$$\mathbb{E}f_\mu(\bar{x}_T) - f^* \leq \frac{6\mu^2 L^2(d+4)^3}{\alpha(T+2c-1)} + \frac{4(2d+9)\sigma^2}{\alpha(T+2c-1)} + \frac{\alpha}{T(T+2c-1)}\Big[c(c-1)\mathbb{E}\left\|x_1 - x^*\right\|^2\Big] \tag{2.22}$$

Where $\bar{x}_T = \frac{2}{T(T+2c-1)}\sum_{t=1}^{T}(t+c-1)x_t$

Proof. Since the gradient-free oracle is an unbiased estimator of f_μ, we will be optimising function f_μ in this proof. Following the proof in Section 3 of Lacoste-Julien et al. (2012) we have (for simplifying the notation in the proof we take $g_t = g_{i_t}(x_t, u_t)$):

$$\begin{aligned} \|x_{t+1} - x^*\|^2 &= \|x_t - \eta_t g_t - x^*\| \\ &= \|x_t - x^*\|^2 + \eta_t^2 \|g_t\|^2 - 2\eta_t (x_t - x^*)^T g_t \end{aligned} \tag{2.23}$$

Taking conditional expectation and using the unbiasedness of the zeroth-order gradient we get:

$$\begin{aligned} \mathbb{E}\big[\, \|x_{t+1} - x^*\|^2 \,\big|x_t, u_t\big] &= \|x_t - x^*\|^2 + \eta_t^2 \mathbb{E}\big[\, \|g_t\|^2 \,\big|x_t, u_t\big] - 2\eta_t (x_t - x^*)^T \nabla f_\mu(x_t) \\ &\overset{(1.7)}{\leq} \|x_t - x^*\|^2 + \eta_t^2 \mathbb{E}\big[\, \|g_t\|^2 \,\big|x_t, u_t\big] - 2\eta_t \big[f_\mu(x_t) - f_\mu(x^*) + \frac{\alpha}{2} \|x_t - x^*\|^2\big] \end{aligned} \tag{2.24}$$

Where the second inequality is the result of strong convexity. Rearranging the above and taking expectation over the randomness of entire sequence of steps we get:

$$2\eta_t \mathbb{E}[f_\mu(x_t) - f_\mu^*] \leq \eta_t^2 \mathbb{E} \|g_t\|^2 + (1 - \alpha\eta_t) \mathbb{E} \|x_t - x^*\|^2 - \mathbb{E} \|x_{t+1} - x^*\|^2 \tag{2.25}$$

Next using the inequality (2.9) in Lemma 6, we can substitute the upper bound for the term $\mathbb{E} \|g_t\|^2$ in equation (2.25). Using the Assumption 1 we will have:

$$\begin{aligned} \mathbb{E}_{u,i} \|g_t\|^2 &\leq 3\mu^2 L^2 (d+4)^3 + 4(d+4)\sigma^2 + 4(d+4) \|\nabla f_\mu(x_t)\|^2 \\ &= 3\mu^2 L^2 (d+4)^3 + A\sigma^2 + B \|\nabla f_\mu(x_t)\|^2 \end{aligned} \tag{2.26}$$

Where we used the variables A and B for simplicity of the notation.

Substituting (2.26) in (2.25), we will get:

$$\begin{aligned} 2\eta_t \mathbb{E}[f_\mu(x_t) - f_\mu^*] &\leq \eta_t^2 \big[3\mu^2 L^2 (d+4)^3 + A\sigma^2 + B\mathbb{E} \|\nabla f_\mu(x_t)\|^2\big] + (1 - \alpha\eta_t) \mathbb{E} \|x_t - x^*\|^2 - \mathbb{E} \|x_{t+1} - x^*\|^2 \\ &\overset{(A.11)}{\leq} \eta_t^2 \big[3\mu^2 L^2 (d+4)^3 + (A+2)\sigma^2 + 4BL\mathbb{E}(f_\mu(x_t) - f_\mu^*))\big] \\ &\quad + (1 - \alpha\eta_t) \mathbb{E} \|x_t - x^*\|^2 - \mathbb{E} \|x_{t+1} - x^*\|^2 \end{aligned} \tag{2.27}$$

Where the second inequality uses the result from Remark 15 which is a consequence of smoothness and Lemma 14. Rearranging the above:

$$2\eta_t (1 - 2\eta_t BL) \mathbb{E}[f_\mu(x_t) - f_\mu^*] \leq 3\eta_t^2 \mu^2 L^2 (d+4)^3 + \eta_t^2 (A+2)\sigma^2 + (1 - \alpha\eta_t) \mathbb{E} \|x_t - x^*\|^2 - \mathbb{E} \|x_{t+1} - x^*\|^2 \tag{2.28}$$

If we choose $\eta_t \leq \frac{1}{4BL}$ then we have: $1 - 2\eta_t BL \leq \frac{1}{2}$. hence:

$$\mathbb{E}[f_\mu(x_t) - f_\mu^*] \leq 1.5\eta_t \mu^2 L^2 (d+4)^3 + \frac{\eta_t}{2}(A+2)\sigma^2 + (\eta_t^{-1} - \alpha) \mathbb{E} \|x_t - x^*\|^2 - \eta_t^{-1} \mathbb{E} \|x_{t+1} - x^*\|^2 \tag{2.29}$$

Define the condition number of function f_μ as $\kappa = \frac{L}{\alpha}$. We choose $\eta_t = \frac{2}{\alpha(c+t)}$, where $c = 8B\kappa$. We will have:

$$\mathbb{E}[f_\mu(x_t) - f_\mu^*] \leq \frac{3\mu^2 L^2 (d+4)^3}{\alpha(t+c)} + \frac{(A+2)\sigma^2}{\alpha(t+c)} + \frac{\alpha}{2}(t+c-2) \mathbb{E} \|x_t - x^*\|^2 - \frac{\alpha}{2}(t+c) \mathbb{E} \|x_{t+1} - x^*\|^2 \tag{2.30}$$

Multiplying inequality (2.30) by $(t+c-1)$, we obtain:

$$\begin{aligned} (t+c-1)\mathbb{E}[f_\mu(x_t) - f_\mu^*] &\leq \frac{3\mu^2 L^2 (d+4)^3 (t+c-1)}{\alpha(t+c)} + \frac{(A+2)\sigma^2 (t+c-1)}{\alpha(t+c)} + \\ &\quad \frac{\alpha}{2}\Big[(t+c-1)(t+c-2) \mathbb{E} \|x_t - x^*\|^2 - (t+c-1)(t+c) \mathbb{E} \|x_{t+1} - x^*\|^2\Big] \end{aligned} \tag{2.31}$$

Summing from $t = 1 \to t = T$, we will obtain the following telescoping sum:

$$\begin{aligned}\sum_{t=1}^{T}(t+c-1)[\mathbb{E}f_\mu(x_t) - f_\mu^*] \leq \frac{3\mu^2L^2(d+4)^3T}{\alpha} + \frac{(A+2)\sigma^2T}{\alpha} + \\ \frac{\alpha}{2}\Big[c(c-1)\mathbb{E}\left\|x_1 - x^*\right\|^2 - (c+T-1)(c+T)\mathbb{E}\left\|x_{T+1} - x^*\right\|^2\Big] \\ \leq \frac{3\mu^2L^2(d+4)^3T}{\alpha} + \frac{(A+2)\sigma^2T}{\alpha} + \frac{\alpha}{2}\Big[c(c-1)\mathbb{E}\left\|x_1 - x^*\right\|^2\Big]\end{aligned} \tag{2.32}$$

Thus using the fact that function f_μ is convex and using the Jensen's inequality yields:

$$\begin{aligned}\mathbb{E}f_\mu\Big(\frac{2}{T(T+2c-1)}\sum_{t=1}^{T}(t+c-1)x_t\Big) - f_\mu^* \leq \frac{6\mu^2L^2(d+4)^3}{\alpha(T+2c-1)} + \frac{2(A+2)\sigma^2}{\alpha(T+2c-1)} + \\ \frac{\alpha}{T(T+2c-1)}\Big[c(c-1)\mathbb{E}\left\|x_1 - x^*\right\|^2\Big]\end{aligned} \tag{2.33}$$

Setting $\bar{x}_T = \frac{2}{T(T+2c-1)}\sum_{t=1}^{T}(t+c-1)x_t$ and substituting A, B and c with their actual values, we will get the statement of the theorem.

$$A = 4(d+4) = \mathcal{O}(d), \qquad B = 4(d+4) = \mathcal{O}(d), \qquad c = 8B\kappa = \mathcal{O}(\kappa d) \tag{2.34}$$

The final asymptotic rate derived from (2.33) is explained in the following remark.

Remark 8. *In equation* (2.33) *by setting* $\mu \leq (Ld^{1.5})^{-1}$ *the first term will be negligible. Furthermore by setting* $\eta_t = \frac{2}{\alpha(c+t)}$*, where* $c = 32(d+4)\kappa$ *and* $\kappa = \frac{L}{\alpha}$*, equation* (2.33) *simplifies to:*

$$\mathbb{E}f_\mu(\bar{x}_T) - f^* \leq \mathcal{O}(\frac{d\sigma^2}{\alpha T}) + \mathcal{O}(\frac{\alpha d^2\kappa^2}{T^2}) \tag{2.35}$$

The dominant term in (2.35) is $\mathcal{O}(\frac{d\sigma^2}{\alpha T})$. This shows that the stochastic zeroth-order algorithm for smooth and strongly convex functions has exactly the same convergence rate as SGD except for a factor which is in the order of dimensionality of the problem. In fact this is the main factor that makes zeroth-order algorithms to scale poorly with dimensionality.

The nice thing about the asymptotic rate in (2.35) is that it is only based on the variance upper bound σ^2. Therefore, we are able to directly use the results we derived in section 2.20 for the variance of the general zeroth-order oracle.

Remark 9. *For optimising the objective function in* (1.1) *we can use a zeroth-order oracle with batch size of* b *and population size of* p*. Particularly if we use the option II oracle defined in* (2.11)*, the asymptotic convergence rate will be:*

$$\mathbb{E}f_\mu(\bar{x}_T) - f^* \leq \mathcal{O}(\frac{d\sigma^2}{\alpha bpT}) + \mathcal{O}(\frac{\alpha d^2\kappa^2}{b^2p^2T^2}) \tag{2.36}$$

Chapter 3

Proposed variants of zeroth-order optimisation methods

The main drawback of the zeroth-order optimisation algorithms is that they scale poorly with the dimensionality of the problem. This could also be observed in the asymptotic rate (2.35) that is, as the dimensionality of the problem increases we need more iterations to reach a certain accuracy. This dependency to dimensionality is linear for strongly convex functions which means that even with moderate number of features like 100 or 1000 (which is normal in Machine Learning applications) our optimisation algorithm will be very slow. This mainly due to the high variance of the zeroth-order oracle because of random search directions which increases with the dimensionality (inequality (2.8)). Moreover, we saw in Section 2.3 that using a population of random directions can help to reduce the variance of this oracle. Intuitively calculating a gradient estimate along many directions gives us more information about the true gradient vector. However, sampling a large population of random directions is computationaly expensive and will in turn make the convergence to happen slower. For example with a population size of p for the oracle (2.1), we have to compute $p + 1$ function evaluations.

Ideally we are seeking for a method which can reduce the variance of the zeroth-order gradient with a smaller population size. In this chapter we introduce three variants of the zeroth-order oracle which aim to reduce the variance. Note that the update steps that we present for each method is for the case with batch size and population size of one but they can be easily generalized to the case with batch size and population size of more than one.

3.1 GD zero-order with momentum

Sutton (1986) states that stochastic gradient descent has problems navigating through ravines, i.e. areas where the function curves much more steeply in one direction than another, which are common around local optima. In these situations, SGD oscillate along the slope of the ravine while only making small progress toward the local optima (Ruder (2016)). Using momentum helps to accelerate the algorithm toward the local optima and dampens the oscillations (Qian (1999)). This is done by adding a fraction β of the past (stochastic) gradients to the current update. Formally our update will be:

$$\begin{aligned} m_t &= \beta m_{t-1} + \eta \nabla f_{i_t}(x_t) \\ x_{t+1} &= x_t - m_t \end{aligned} \tag{3.1}$$

Ruder (2016) provides a nice analogy for momentum, it is like pushing a ball down a hill, the ball accumulates momentum on its way and becomes faster and faster. This also illustrated in Figure 3.1. An accelerated version of momentum is Nesterov Accelerated Gradient (NAG) introduced in NESTEROV (1983). The idea is that as we know that our momentum term is going to roughly move our parameters to $x_t - \beta m_{t-1}$, then we can compute the stochastic gradient at $x_t - \beta m_{t-1}$. This way we can effectively look ahead and compute the gradient with respect to approximate future position of our parameters(Ruder (2016)). This idea becomes clear by looking at the picture in Figure 3.2. The

[1]Figure from Du (2019)

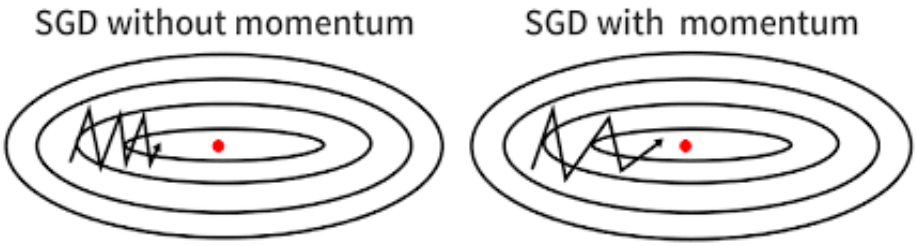


Figure 3.1: Effect of momentum in SGD [1]

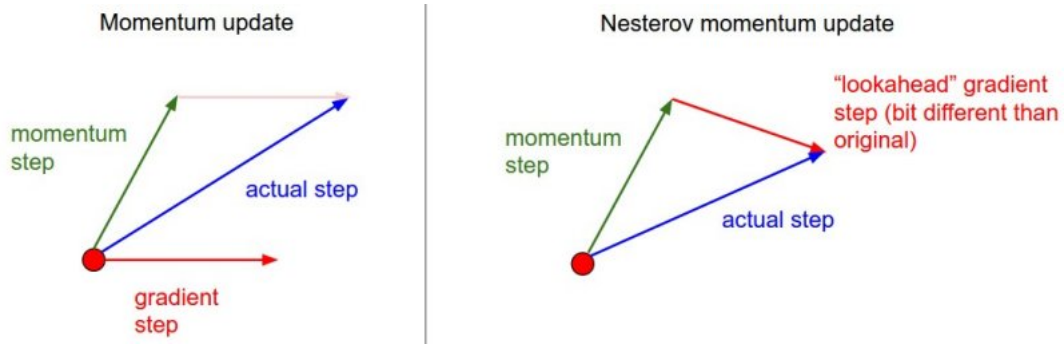


Figure 3.2: NAG update (source:Stanford CS231n class [2])

update of NAG is as follows:

$$\begin{aligned} m_t &= \beta m_{t-1} + \eta \nabla f_{i_t}(x_t - \beta m_t) \\ x_{t+1} &= x_t - m_t \end{aligned} \tag{3.2}$$

In zeroth-order setting we have also a similar situation due to the new source of randomness, i.e. random directions. With Sampling random directions at each step, there is a chance that the algorithm goes into directions which is not along the steepest descent direction. Therefore the algorithm will be oscillating along random directions, progressing slowly toward the optima. Therefore momentum gives us a short-term memory with which we can partially store the information from the random directions we picked in the previous iterations. This is particularly useful because the past gradient estimates were calculated along different directions and hence we will have gradient estimates along different directions coming with no additional cost. This will in turn dampen the oscillations and makes the progress of the algorithm toward local optima faster. In our experiments we observed that using the NAG update in the oracle outperforms the regular momentum update (equation (3.1)) in zeroth-order setting. In what follows we call this method as `(S)GD_ZO_momentum` with the update step:

$$\begin{aligned} m_t &= \beta m_{t-1} + \eta g_{i_t}(x_t - \beta m_t, u_t) \\ x_{t+1} &= x_t - m_t \end{aligned} \tag{3.3}$$

The value of momentum should be between zero and one ($0 < \beta < 1$). If $\beta = 0$ we will get back to `SGD_ZO` and if it is equal to or larger than one then the memory m_t will explode and the method diverges. So β is a hyper-parameter that should be tuned. It is desirable to tune β and η simultaneously to get the best possible pair of parameters.

3.2 GD zero-order with averaging

As stated earlier in Section 3.1, in zeroth-order setting momentum helps to reuse the gradient estimates of the past iterations. Although these extra gradient estimates are old, they turn out to be useful. Moreover, with momentum we have an exponential weighting scheme and we give less weight to

[2] http://cs231n.github.io/neural-networks-3/

very old estimates. An alternative idea is to use a uniform averaging scheme. To avoid reusing very old estimates we can keep the last K gradient estimates computed in the previous iterations in memory and add their average to the current gradient estimate. This is somehow similar to the SAGA algorithm introduced in Defazio et al. (2014) except that here we are sampling random directions from a continuous distribution and hence we do not have a fixed set of outputs. Therefore we only keep the last K gradient estimates. We call this algorithm as `(S)GD_ZO_avg_win`. The update step of this algorithm is as follows:

$$\begin{aligned} m_t &= g_{i_t}(x_t, u_t) + \frac{1}{k}\sum_{k=1}^{K} g_{i_k}(x_{t-k}, u_{t-k}) \\ x_{t+1} &= x_t - \eta m_t \end{aligned} \tag{3.4}$$

We call the memory in which we keep the last K gradient estimates as "window" since we have to update the memory in each iteration and it is analogous to moving a window of size K on the iterates. This window size is a hyper-parameter and it should be tuned. The range of possible values for K is somehow open and one can arbitrarily choose a large value for K. However, having a large K means that we are reusing very old information which is most probably irrelevant. There is no clear way to find a suitable range of integers to search for K.

3.3 Keeping a population of random directions in memory

The idea in this algorithm is to directly reuse the sampled random directions in the previous iterations (not the previous calculated estimates). Specifically, The idea is to sample multiple random directions only in a few iterations and then in between these costly iterations we use the information from the previous population of random directions to compute the gradient estimate. We can formalize this scheme as follows: every k iterations we compute a zeroth-order gradient with a population of size p i.e. we independently sample p random directions and hence compute p gradient estimates. Then we use the average of these gradient estimates as the update step (a costly but good step). In the other iterations we only sample one random direction to calculate a gradient estimate with population size of one. We also use the random directions that were picked in the previous costly step to compute a (noisy) gradient estimate. Then we pick the weighted average of these two estimates as our update step. We refer to this algorithm as `(S)GD_ZO_pop_mem` and its update step is:

$$\begin{aligned} \text{assume: } s >> 1, k \mid s: \quad m_s &= \frac{1}{p}\sum_{j=1}^{p} g_{i_s}(x_s, u_{s,j}) \\ x_{s+1} &= x_s - \eta m_s \end{aligned} \tag{3.5}$$

$$\begin{aligned} \text{for } s+1 \le t < s+k: \quad m_t &= \frac{1}{p+(t-s)}\Big(\sum_{j=1}^{p} \frac{f(x_s+\mu u_{s,j}) - f(x_t)}{\|x_s + \mu u_{s,j} - x_t\|}(x_s + \mu u_{s,j} - x_t) + (t-s) g_{i_t}(x_t, u_t)\Big) \\ x_{t+1} &= x_t - \eta m_t \end{aligned} \tag{3.6}$$

The update frequency of the population k is a hyper-parameter in this method. Similar to method in Section 3.2, tuning this parameter is hard as k could be arbitrarily large. In practice usually small values for k work better since with a large k we will be reusing old points ,which may be far from the current point, for estimating the gradient .

In our implementation we made a slight change to the update in (3.6). In order to have more fresh random directions, in each of the cheap steps we replace one of the old random directions from the population with the new random direction sampled in the current step. This slight change empirically showed to improve the convergence speed of the method.

Chapter 4

Experiments

We implemented the zeroth-order stochastic optimisation with the various update steps introduced in Chapter 3. We provide a simulation framework in Python for zeroth-order optimisation. In this chapter we present some numerical experiments to illustrate the difference of the proposed zeroth-order oracles in terms of the convergence speed. The data sets we used for our experiments have a few number of features. The reason is that we wanted to run our experiments in a reasonable amount of time as the zeroth-order method scale poorly with the dimensionality of the problem. We performed all of our experiements in both the non-stochastic (when function f is fixed, so there is no stochasticity in terms of the choice of f_i) and stochastic setting. In the stochastic setting we also investigate the effect of having different batch-sizes. Finally in Section 4.5.2 we will discuss about some tricks to use the population of random directions more effectively.

4.1 Experimental setup

As stated in Section 1.4.1 we used the regularized logistic regression with the objective function:

$$\mathcal{L}(x) = \frac{-1}{n} \sum_{i=1}^{n} b_i \ln \sigma(a_i^T x) + (1 - b_i) \ln \left[1 - \sigma(a_i^T x)\right] + \frac{\lambda}{2} \|x\|^2 \tag{4.1}$$

Where $a_i \in \mathbb{R}^d$ are the feature vectors and $b_i \in \{0, 1\}$ are data labels. The regularization parameter is set to $\lambda = \frac{1}{n}$ according to Bottou (2010). Also the smoothing parameter is set to $\mu = \frac{1}{Ld^{1.5}}$ according to Remark 18.

Hardware. Experiments were run on an Ubuntu 16.04 machine with a 24 cores processor Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz.

4.2 Data-set

We used the small 8×8 hand-written digits data-set from UCI machine learning repository (Dua and Graff (2017)). These data samples have $d = 64$ features and we used a sub-sample of the data with size $n = 1437$. We used this data-set in a binary classification task to separate digits smaller than five and digits bigger or equal to five. Figure 4.1 shows a sample of these hand-written digits.

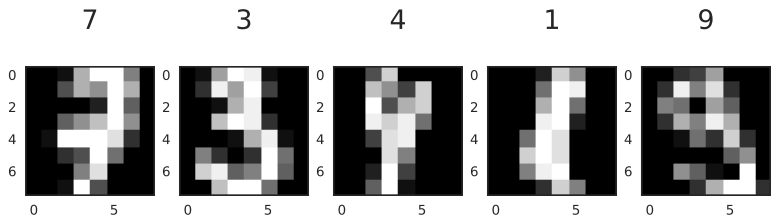


Figure 4.1: 8×8 hand-written digits

4.3 Implementation

We developed a framework for implementing regularized logistic regression with various optimisers using Python3 and numpy library Jones et al. (2001–). This is a high-level optimisation and it is not optimised for clock-wise speed. In order to have a fair comparison between the proposed methods we ran the different algorithms starting from a fix initial point and compared them after running for the same number of function evaluations.

4.4 Baselines

We implemented our own version of (S)GD to have a baseline to compare zeroth-order methods with. To do so, we run (S)GD on our data-sets for ample number of iterations to get a baseline loss. Then in our plots we show the baseline loss as a dashed black line so that we can see how close the zeroth-order methods can get to the baseline loss. The reason that we implemented our own version of (S)GD is that we were not only interested to see how close the zeroth-order algorithms get to optimum but also we were interested to compare the speed of convergence of zeroth-order methods with (S)GD.

4.5 Results

In this section we present the result of our experiments. Each of the experiments in this section are done for two scenario:

- function f is fixed, hence there is no stochastisity in terms of choosing random functions. In this case we compare our proposed methods with gradient descent (GD) as a baseline. we study the convergence of each method using a fixed step-size which we tune separately for each method.
- function f is a sum of n functions as in (1.1). In this case we compare our proposed methods with stochastic gradient descent (SGD) as a baseline. For the stochastic methods we use the time variant step-size suggested by Bottou (2012):

$$\eta_t = \frac{\eta_0}{1 + \eta_0 \lambda t} \tag{4.2}$$

  where η_0 is a tunable parameter. Same as the non-stochastic case, we tune η_0 for each method separately.

4.5.1 Comparison of the proposed methods

We start by comparing the methods introduced in Section 3 with the basic zeroth-order oracle introduced in (2.1). We used the same population size (and batch size for the stochastic case) for all of the methods. For the methods introduced in Section 3.1 and 3.2, we also tuned the momentum value

and window size respectively. Figure 4.2 shows the convergence of these different methods in the non-stochastic case and with population size of 10. Note that the x axis in this figure indicates the number of function evaluations divided by n (number of data points). One function evaluation is equivalent to the computation of $f(x)$.

We can observe that all the three methods introduced in Section 3 improve over the basic zeroth-order oracle. Especially using the Nesterov momentum (NAG) in zeroth-order oracle significantly improves the speed of convergence in the non-stochastic setting.

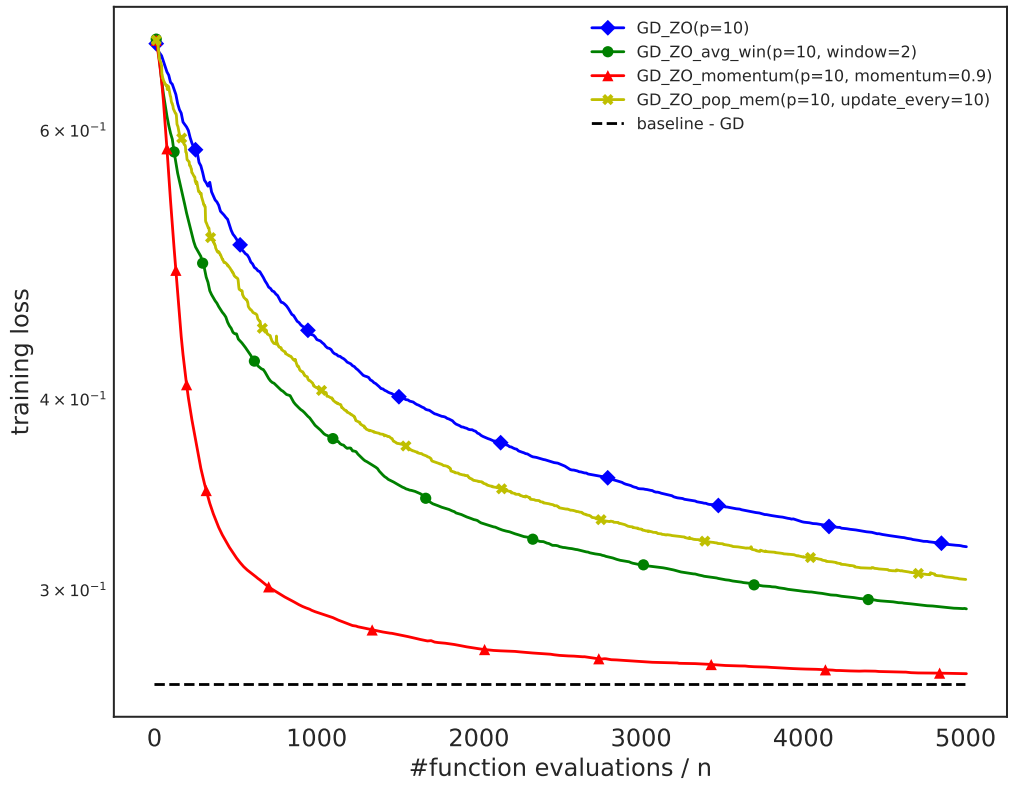
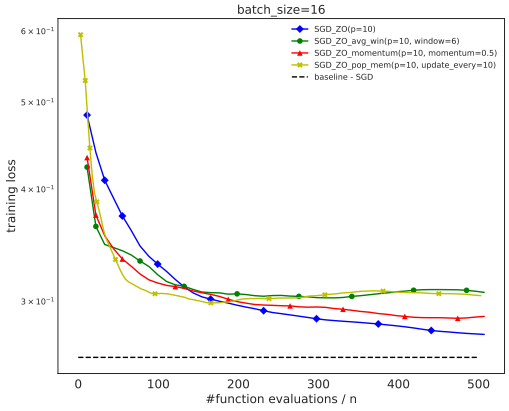


Figure 4.2: Comparison of different zeroth-order methods in the non-stochastic case

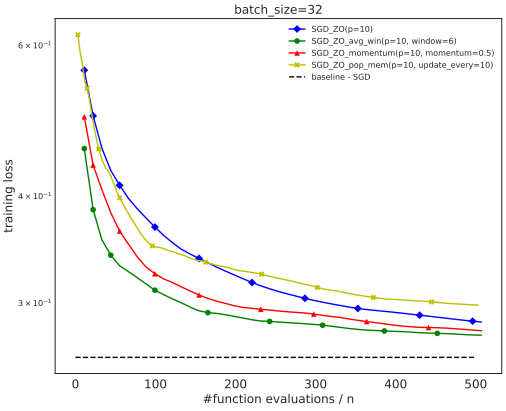
Figure 4.3 compares the convergence of these methods in the stochastic case. We can observe that in this case methods (3.2) and (3.4) are still better than the basic oracle, however the difference is less significant especially with smaller batch sizes. One can observe that with batch size 16 there is almost no difference between `SGD_ZO_momentum` and `SGD_ZO_avg_win` with the basic oracle `SGD_ZO`. The method `SGD_ZO_pop_mem` works poorly with small batch size however, it gets better by increasing the batch size and we can see that with batch size 64 it is almost as good as `SGD_ZO_momentum` and `SGD_ZO_avg_win`.

4.5.2 A note about how to use the population

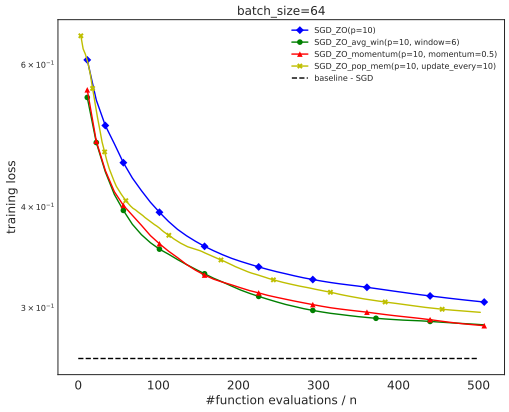
So far the way we were using the population was by uniform averaging (same as in oracles (2.10) and (2.11)). However one can observe that uniformly averaging the population is not efficient. The reason is that when we sample p random directions from a $\mathcal{N}(0, I)$, these random directions are not equally good. Therefore it makes sense to use a weight scheme to favor "better" directions i.e. the ones which are more aligned with the true gradient direction. Hansen (2016) proposes to rank the population based on how much they decrease the function value. The idea is that random directions for which



(a) batch size 16



(b) batch size 32



(c) batch size 64

Figure 4.3: Comparison of the proposed methods in the stochastic case

the difference $|f(x+\mu u) - f(x)|$ is higher should be favored as they are more aligned with the true gradient direction. Note that here we rank based on the absolute value of the difference, the reason is that directions which are completely in the opposite direction of gradient are also interesting as the direction will be fixed by the sign of $f(x+\mu u) - f(x)$.

Hansen (2016) proposed to select a subset of the better directions and use them to compute the gradient estimate with a weighted average. In this section we explored two schemes for using the population to compute the gradient estimate:

- Ranking the random directions based on the value $\frac{|f(x+\mu u)-f(x)|}{\|u\|}$ and selecting the gradient estimate associated with the best random direction. Note that we are interested in the direction of the random vector u, not its norm. Therefore it makes sense to normalize the difference of the function values by $\|u\|$. Formally assume that $u_{i:p}$ is the i-th best random direction in the population, i.e. we have : $\frac{|f(x+\mu u_{1:p})-f(x)|}{\|u_{1:p}\|} \leq ... \leq \frac{|f(x+\mu u_{p:p})-f(x)|}{\|u_{p:p}\|}$. Then the gradient estimate would be:

$$G_t = \frac{|f(x_t + \mu u_{1:p}) - f(x_t)|}{\|u_{1:p}\|} \tag{4.3}$$

- Doing a weighted average over the whole population with the weights being proportional to the value $\frac{|f(x+\mu u)-f(x)|}{\|u\|}$. Formally our weighted average scheme would be:

$$\text{set of random directions } \{u_1, ..., u_p\}$$
$$G_t = \frac{1}{\sum_{j=1}^{p} \frac{|f(x_t+\mu u_j)-f(x_t)|}{\|u_j\|}} \sum_{j=1}^{p} \Big(\frac{|f(x_t + \mu u_j) - f(x_t)|}{\|u_j\|} \Big) g(x_t, u_j) \tag{4.4}$$

We compared these two schemes against the uniform average scheme (`(S)GD_ZO`) and report the results in Figures 4.4 and 4.5. We run each algorithm five times to observe the variance due to the random directions. Figure 4.4 shows that in the non-stochastic setting both selecting based on the best random direction or the weighted average in (4.4) are better than uniform averaging. However, it can be seen that selecting based on the best random direction introduces a high variance.

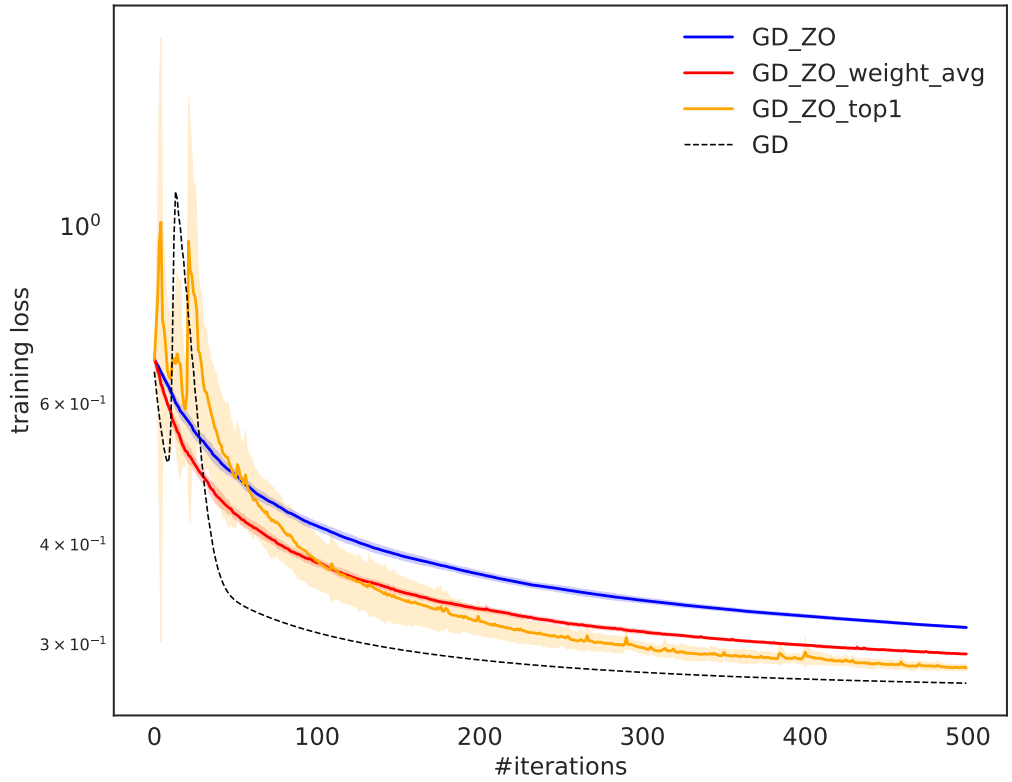
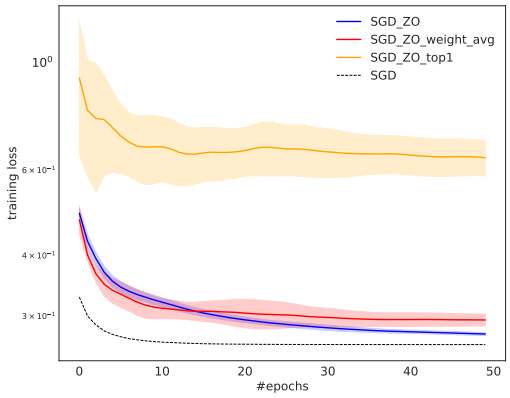
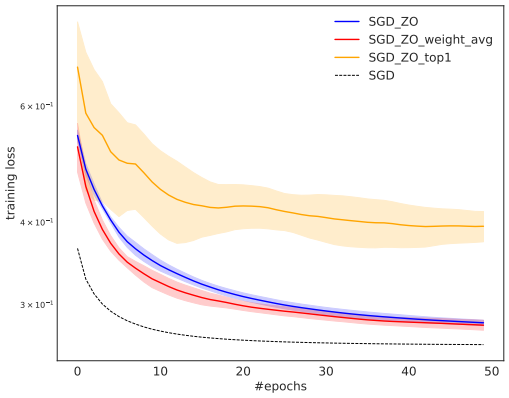


Figure 4.4: how to use the population? - non-stochastic case

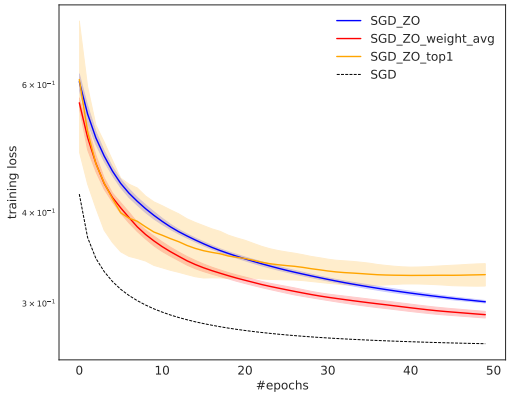
Figure 4.5 shows that selecting based on the best random direction performs poorly in the stochastic case. One possible reason could be that in the stochastic case we are favoring the random directions that are more aligned with the direction of the gradient of f_i (or the direction of the gradient of a batch of random functions). However this direction is not necessarily aligned with the full gradient direction and therefore estimating the gradient only along the best direction may introduce a large amount of noise. Looking into the graphs in Figure 4.5 also confirms this hypothesis, by increasing the batch size the method based on selecting the best random direction gets better. This is because the gradient of a larger batch of functions is closer to the full gradient. Nevertheless, the weighted average scheme is still better than uniform averaging especially with larger batch sizes.



(a) batch size 16



(b) batch size 32



(c) batch size 64

Figure 4.5: how to use the population? - stochastic case

4.5.3 Efficiency of gradient vs zeroth-order gradient

The results in Section 4.5.2 shows that progressing in the direction of the gradient is much more effective for optimising a convex function, i.e. the rate of reduction of the loss is faster for a first-order method compared to a zeroth-order method. This observation is as expected, however it would be interesting to look into this observation more quantitatively. In other words we want to see how many first-order and zeroth-order gradient calls are required to reach a certain value of the loss function. Moreover, this will give us another criteria to compare our proposed variants of zeroth-order algorithms introduced in Section 3 with the basic zeroth-order method (`(S)GD_ZO`). Figure 4.6 shows an equivalence between number of function evaluations and gradient computations in the non-stochastic case. From this figure we can also compute the average ratio of function evaluations being equivalent to one gradient. For these experiments we only considered the zeroth-order method with momentum as it seems to be the best method among others according to the results in Section 4.5.1. Figure 4.6 confirms the effectiveness of momentum in the non-stochastic setting, we can see that using momentum we need less number of function evacuations to make a step equivalent to the gradient step.

Figure 4.7 shows the equivalence between number of function evaluations and number of gradient computations in the stochastic setting. Again we can observe the superiority of momentum method in this figure and this difference becomes more significant with larger batch sizes.

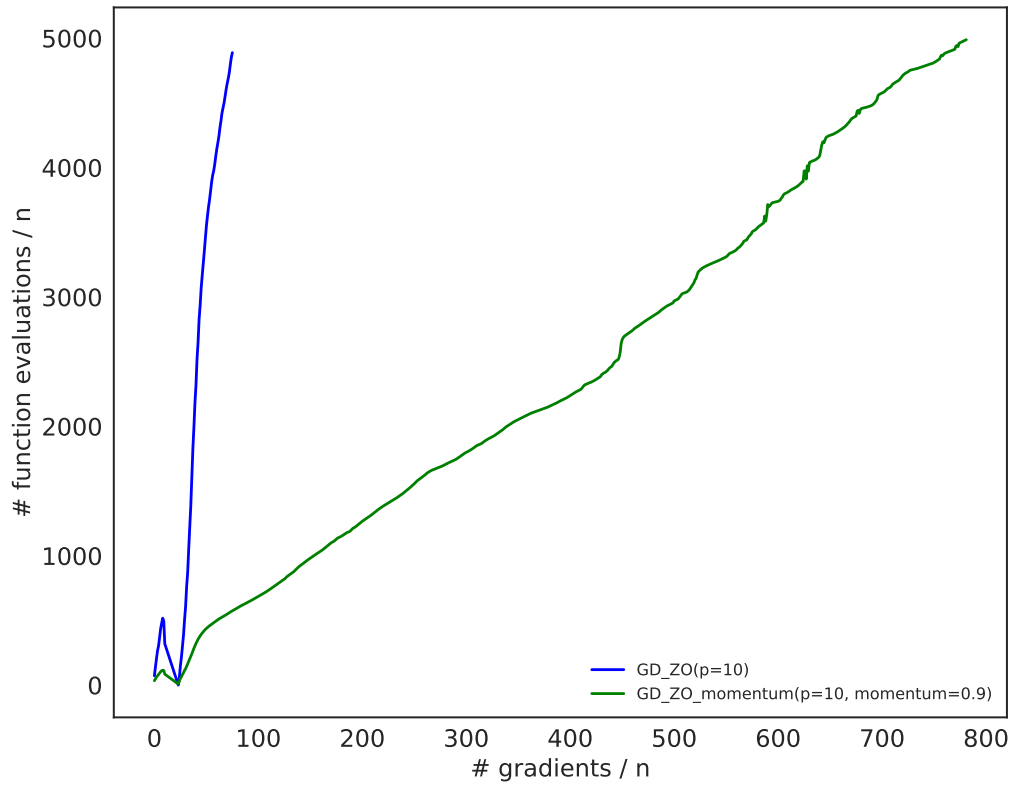
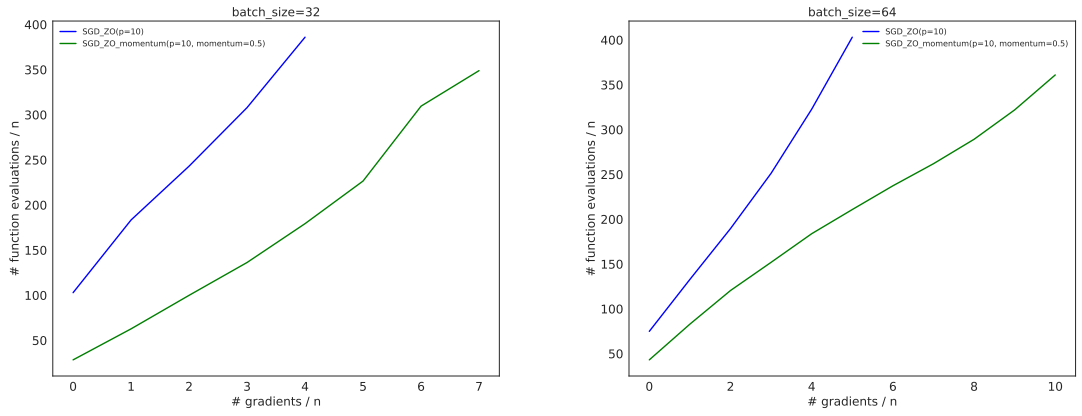


Figure 4.6: how many function evaluations is equivalent to a gradient step in non-stochastic setting. With `GD_ZO` on average 54.5 function evaluations is equivalent to one full gradient whereas for `GD_ZO_momentum` this ratio is 6.4.



(a) batch size 32. ratio for `SGD_ZO` = 84.5 for `SGD_ZO_momentum` = 37.1

(b) batch size 64. ratio for `SGD_ZO` = 66.6 for `SGD_ZO_momentum` = 36.1

Figure 4.7: how many function evaluations is equivalent to a gradient step in stochastic setting. The ratio of number of function evaluations over number of gradients is represented for each batch size.

4.6 Discussion

As we proved in Section 1.3, zeroth-order methods are d times slower than the gradient methods in the strongly convex setting. However it is difficult to find an algorithm that can theoretically improve this factor of d in the convergence rate of the zeroth-order methods. Nevertheless, our experiments shows that using the information from the previous calculated estimates do help to make the convergence happen faster. This is due to the fact that in the smooth and strongly convex setting, the function values does not change drastically. Therefore the information that we have from the random directions sampled in the previous "not too old" iterations are still valid and could be used in the current iteration.

We used the same Python implementation for comparing the methods and tried to be fair in our experiments. However, it could be argued that we didn't invest enough tuning for the `(S)GD_ZO_pop_mem` method as we did not tune the update frequency of sampling fresh population. In practice it turns out that tuning the momentum in `(S)GD_ZO_momentum` is less cumbersome compared to other two methods introduced in Chapter 3 since the range of possible values for momentum is limited. After all, the main goal of these set of experiments was to show that using the information from previous iterations helps to speed up the convergence of the zeroth-order methods without additional cost in terms of the number of function evaluations.

Conclusion and future work

We provide the first analysis of stochastic zeroth-order optimisation (SGD_ZO) in smooth and strongly convex setting which is based on a time variant step size and a $t+1$ averaging of the iterates. We show that SGD_ZO reaches accuracy $\mathcal{O}\big(\frac{d}{T}\big)$ after T iterations. This asymptotic rate clearly shows the dependency of the convergence rate to the dimensionality of the problem. We also derived upper bounds for the variance of the zeroth-order oracle with batch size and population size of bigger than one. Our variance bounds shows that SGD_ZO enjoys a linear improvement in convergence rate by increasing the batch size and population size.

We empirically showed that zeroth-order methods which use information from the previous iterations have a faster convergence rate (e.g. SG_ZO_momentum). A future line of work could be to try SG_ZO_momentum on a common application of gradient free schemes i.e. generation of adversarial examples from a black-box model. Liu et al. (2018) proposed a method called ZO_SVRG and showed in their experiments that it performed better than SGD_ZO in a black-box attack to a DNN trained on the MNIST hand-written digits data-set. It would be interesting to try SGD_ZO_momentum in the same black-box attack problem and compare its performance against SVRG_ZO.

We also argued that random directions are not equally good and we are interested in the random directions which are more aligned with the gradient direction. Our experiments also confirms this fact, as we saw that favoring the "better" directions by using a weighted averaging makes the convergence to happen faster. Hansen (2016) uses the same argument in a similar problem. They further argue that they can update the covariance matrix of the Gaussian (which is initialized with an identity matrix) by estimating it from the better directions. It will be interesting to try this scheme in our zeroth-order methods i.e. updating the covariance matrix of the Gaussian distribution in SGD_ZO. This way we can go toward the distributions which generate good directions with higher probability.

Appendices

Appendix A

Useful lemmas

Lemma 10. *Given two vectors $a, b \in \mathbb{R}^d$, we have:*

$$2\langle a, b\rangle \leq \|a\|^2 + \|b\|^2 \tag{A.1}$$

Proof.

$$0 \leq \|a - b\|^2 \leq \|a\|^2 + \|b\|^2 - 2\langle a, b\rangle \tag{A.2}$$

Lemma 11. *For any vectors $a_1, a_2, ..., a_k \in \mathbb{R}^d$, we have:*

$$\left\|\sum_{i=1}^{k} a_i^2\right\|^2 \leq k\sum_{i=1}^{k} \|a_i\|^2 \tag{A.3}$$

Proof. This is true for $k = 1$. By induction we have:

$$\begin{aligned}\left\|\sum_{i=1}^{k} a_i^2\right\|^2 &\leq (k-1)\sum_{i=1}^{k-1} \|a_i\|^2 + \|a_k\|^2 + 2\sum_{i=1}^{k-1} \langle a_k, a_i\rangle \\ &\leq (k-1)\sum_{i=1}^{k-1} \|a_i\|^2 + k\|a_k\|^2 + \sum_{i=1}^{k-1} \|a_i\|^2 \\ &= k\sum_{i=1}^{k} \|a_i\|^2\end{aligned} \tag{A.4}$$

Where the first inequality uses the induction hypothesis and the second inequality uses Lemma 10.

Lemma 12. *Given L-smooth functions $f_i : \mathbb{R}^d \to \mathbb{R}, i \in [n]$, the function f defined as:*

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{A.5}$$

is L-smooth.

Proof. from Definition 2 and by linearity of gradient we have:

$$\begin{aligned}f(y) &= \frac{1}{n}\sum_{i=1}^{n} f_i(y) \\ &\leq \frac{1}{n}\left[f_i(x) + \langle \nabla f_i(x), y - x\rangle + \frac{L}{2}\|x - y\|^2\right] \\ &= f(x) + \langle \nabla f(x), y - x\rangle + \frac{L}{2}\|x - y\|^2\end{aligned} \tag{A.6}$$

Lemma 13. *[Bubeck (2014)] Let f be L-smooth, then for any $x, y \in \mathbb{R}^d$ we have:*

$$f(x) - f(y) \leq \nabla f(x)^T (x - y) - \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|^2 \tag{A.7}$$

Proof. Let $z = y - \frac{1}{L}(\nabla f(x) - \nabla f(y))$. Then we have:

$$\begin{aligned} f(x) - f(y) &= f(x) - f(z) + f(z) - f(y) \\ &\leq \nabla f(x)^\top (x - z) + \nabla f(y)^\top (z - y) + \frac{L}{2} \|z - y\|^2 \\ &= \nabla f(x)^\top (x - y) + (\nabla f(x) - \nabla f(y))^\top (y - z) + \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|^2 \\ &= \nabla f(x)^\top (x - y) - \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|^2 \end{aligned} \tag{A.8}$$

Lemma 14. *[Bubeck (2014)] Suppose that $f(x)$ is L-smooth and x^* is the optimum point of $f(x)$, i.e. $\nabla f(x^*) = 0$. Then we will have the following inequality as a result of smoothness:*

$$\mathbb{E}_i[\|\nabla f_i(x) - \nabla f_i(x^*)\|^2] = \frac{1}{n} \sum_{i=1}^{n} \|\nabla f_i(x) - \nabla f_i(x^*)\|^2 \leq 2L(f(x) - f(x^*)) \tag{A.9}$$

Proof. Let $g_i(x) = f_i(x) - f_i(x^*) - \nabla f_i(x)^\top (x - x^*)$. Note that $g_i(x^*) = 0$ and $\nabla g_i(x^*) = 0$. Using the Lemma 13 we have $-g_i(x) \leq -\frac{1}{2L} \|\nabla g_i(x)\|^2$ which can be equivalently written as:

$$\|\nabla f_i(x) - \nabla f_i(x^*)\|^2 \leq 2L(f_i(x) - f_i(x^*) - \nabla f_i(x^*)^\top (x - x^*)) \tag{A.10}$$

Taking expectation with respect to i and observing that $\mathbb{E} \nabla f_i(x^*) = \nabla f(x^*) = 0$ yields the result.

Remark 15. *Assume that function f_i is L-smooth and $\mathbb{E} \|\nabla f_i(x) - \nabla f(x)\|^2 \leq \sigma^2$. As a consequence of Lemma 14 we have:*

$$\mathbb{E}_i \left[\|\nabla f_i(x)\|^2 \right] \leq 4L(f(x) - f(x^*)) + 2\sigma^2 \tag{A.11}$$

Proof. From Lemma 14 we have:

$$\begin{aligned} \mathbb{E}_i \left[\|\nabla f_i(x)\|^2 \right] &= \mathbb{E}_i \left[\|\nabla f_i(x) - \nabla f_i(x^* + \nabla f_i(x^*)\|^2 \right] \\ &\leq 2\mathbb{E}_i \|\nabla f_i(x) - \nabla f_i(x^*\|^2 + 2\mathbb{E}_i \|\nabla f_i(x^*)\|^2 \\ &\leq 4L(f(x) - f(x^*)) + 2\mathbb{E}_i \|\nabla f_i(x^*) - \nabla f(x^*)\|^2 \\ &\leq 4L(f(x) - f(x^*)) + 2\sigma^2 \end{aligned} \tag{A.12}$$

Lemma 16. *Rakhlin et al. (2011)Let f be α-strongly convex and $\mathbb{E} \|g_t\|^2 \leq G^2$ where g_t is the stochastic gradient at time t. Then if we pick $\eta_t = \frac{1}{\alpha t}$, it holds for any T that:*

$$\mathbb{E} \|x_T - x^*\|^2 \leq \frac{4G^2}{\alpha^2 T} \tag{A.13}$$

and for the first iterate x_0 we have:

$$\mathbb{E} \|x_0 - x^*\| \leq \frac{2G}{\alpha} \tag{A.14}$$

Appendix B

Alternative analysis of the stochastic zeroth-order algorithm under strong convexity

We will introduce an alternative analysis of the stochastic zeroth-order algorithm for smooth and strongly convex functions. The main difference of this analysis compared to analysis in Section 2.4 is that here we assume a constant step size η and we prove the convergence of the last iterate x_T. Similar to Section 2.4 we mainly use the Assumption 1 in this analysis. The main result is presented in Theorem 17. The resulting inefficient convergence rate shows the importance of time variant step size and t-averaging of the iterates in the analysis of stochastic zeroth-order algorithm.

Theorem 17. *Suppose that we have the objective function defined in* (1.1) *and each f_i is L-smooth and f is α-strongly convex. Further suppose that for optimising f we will use SGD introduced in Algorithm 2 except that we use the stochastic zeroth-order oracle in* (2.1) *for our updates. By choosing $\eta \leq \frac{1}{8(d+4)L}$ after T iterations of the algorithm we will have:*

$$\mathbb{E}\left\|x_T - x_*\right\|^2 \leq (1-\eta\alpha)^T \mathbb{E}\left\|x_0 - x_*\right\|^2 + \frac{\eta(3\mu^2 L^2 (d+4)^3 + 8(d+4)\sigma^2)}{\alpha} \tag{B.1}$$

We will follow the proof in Section 3.1 of Mansel Gower et al. (2019).

Proof. Let $r_t = x_t - x^*$. From the update step we have:

$$\begin{aligned} \left\|r_{t+1}\right\|^2 &= \left\|x_t - x^* - \eta g_{i_t}(x_t, u_t)\right\|^2 \\ &= \left\|r_t\right\|^2 - 2\eta \left\langle r_t, g_{i_t}(x_t, u_t)\right\rangle + \eta^2 \left\|g_{i_t}(x_t, u_t)\right\|^2 \end{aligned} \tag{B.2}$$

Now taking expectation over u_t and i_t and using the strong convexity definition in 1.7 we will have:

$$\begin{aligned} \mathbb{E}_{u_t, i_t}\left\|r_{t+1}\right\|^2 &= \left\|r_t\right\|^2 - 2\eta \left\langle r_t, \nabla f_\mu(x_t)\right\rangle + \eta^2 \mathbb{E}_{u,i_t}\left\|g_{i_t}(x_t, u_t)\right\|^2 \\ &\overset{(1.7)}{\leq} (1-\eta\alpha)\left\|r_t\right\|^2 - 2\eta[f_\mu(x_t) - f_\mu(x^*)] + \eta^2 \mathbb{E}_{u_t,i_t}\left\|g_{i_t}(x_t, u_t)\right\|^2 \end{aligned} \tag{B.3}$$

Where the second inequality comes from the definition of strong convexity. Now we have to apply the upper bound on $\mathbb{E}_{u_t,i_t}\left\|g_{i_t}(x_t, u_t)\right\|^2$ from Lemma 2.8. We have:

$$\mathbb{E}_{u_t,i_t}\left\|g_{i_t}(x_t, u_t)\right\|^2 \overset{(2.9)}{\leq} 3\mu^2 L^2 (d+4)^3 + 4(d+4)\mathbb{E}_{i_t}\left\|\nabla f_{\mu,i_t}(x_t)\right\|^2 \tag{B.4}$$

From now on to simplify our notation we assume that $3\mu^2 L^2 (d+4)^3 = C$. Using the Remark 15 we have:

$$\mathbb{E}_{u_t,i_t}\left\|g_{\mu,i_t}(x_t, u_t)\right\|^2 \leq C + 16(d+4)L[f_\mu(x_t) - f_\mu(x^*)] + 8(d+4)\sigma^2 \tag{B.5}$$

Hence we will have:

$$\mathbb{E}_{u_t,i_t} \|r_{t+1}\|^2 \leq (1-\eta\alpha) \|r_t\|^2 + 2\eta(8(d+4)\eta L - 1)[f_\mu(x_t) - f_\mu(x^*)] + \eta^2(C + 8(d+4)\sigma^2) \quad \text{(B.6)}$$

If we choose step size $\eta \in (0, \frac{1}{8(d+4)L}]$ then the second term will be smaller or equal to zero:

$$\mathbb{E}_{u_t,i_t} \|r_{t+1}\|^2 \leq (1-\eta\alpha) \|r_t\|^2 + \eta^2(C + 8(d+4)\sigma^2) \quad \text{(B.7)}$$

Finally we take expectation over the whole randomness of $i_1, i_2, ...$ and $u_1, u_2, ...$:

$$\mathbb{E} \|r_{t+1}\|^2 \leq (1-\eta\alpha)\mathbb{E} \|r_t\|^2 + \eta^2(C + 8(d+4)\sigma^2) \quad \text{(B.8)}$$

By recursively applying the above and summing up the geometric series from $t = 0$ to $t = T-1$ we will have:

$$\begin{aligned} \mathbb{E} \|r_t\|^2 &\leq (1-\eta\alpha)^t \mathbb{E} \|r_0\|^2 + \sum_{j=0}^{T-1} (1-\eta\alpha)^j (\eta^2(C + 8(d+4)\sigma^2)) \\ &\leq (1-\eta\alpha)^T \mathbb{E} \|r_0\|^2 + \frac{\eta(C + 8(d+4)\sigma^2)}{\alpha} \end{aligned} \quad \text{(B.9)}$$

Following the analysis in Section A.1 of Mansel Gower et al. (2019) we can find the number of steps required to have $\mathbb{E} \|x_T - x^*\|^2 \leq \epsilon$. By choosing step size $\eta = \min\{\frac{1}{8(d+4)L}, \frac{\epsilon\alpha}{2(C+8(d+4)\sigma^2)}\}$ we will have:

$$T \geq max\{\frac{8(d+4)L}{\alpha}, \frac{2(C + 8(d+4)\sigma^2)}{\epsilon\alpha^2}\} log(\frac{2 \|x_0 - x^*\|^2}{\epsilon}) \quad \text{(B.10)}$$

Remark 18. *If we choose $\mu < (Ld^{1.5})^{-1}$ and for small enough ϵ we choose $\eta = \frac{\epsilon\alpha}{2(C+8(d+4)\sigma^2)}$, we will have accuracy ϵ after $\mathcal{O}(\frac{d\sigma^2}{\epsilon\alpha^2} log(\frac{\|x_0-x^*\|^2}{\epsilon}))$ iterations.*

By comparing the derived rate in Remark 18 with the asymptotic rate in (2.35), we can see that here we have an extra logarithmic term. Moreover, the step size being dependent to the accuracy seems to be a strong restriction. This means that for reaching more accuracy (smaller ϵ), we have to reduce the step size accordingly.

Appendix C

Zeroth-order stochastic variance reduced gradient

Liu et al. (2018) provides a theoretical analysis for ZO_SVRG in the non-convex setting which shows that it achieves a $\mathcal{O}\big(\frac{d}{T}+\frac{1}{b}\big)$ convergence rate where b is the batch size. They also discuss to use a population of random directions to mitigate the $\mathcal{O}\big(\frac{1}{b}\big)$ error term. These results motivates us to analyze ZO_SVRG in the smooth and strongly convex setting. Similarly, our analysis also shows that there is an extra error term that prevents ZO_SVRG to reach a linear convergence in the strongly convex setting. We also discuss about using batch size and population size to mitigate this error term.

The idea of SVRG was initially introduced by Johnson and Zhang (2013). We will first take a brief look at this algorithm in the zeroth-order setting. At every m iterations of the ZO_SGD we keep a snapshot point $\tilde{x}$ and the average gradient $\tilde{g}$ computed at point $\tilde{x}$. For now assume that the population size is one.

$$\tilde{g}=\frac{f(\tilde{x}+\mu\tilde{u})-f(\tilde{x})}{\mu}\tilde{u}=\frac{1}{n}\sum_{i=1}^{n}g_i(\tilde{x},\tilde{u}) \tag{C.1}$$

Computation of $\tilde{g}$ requires one full pass over the data which is equivalent to $n+1$ function evaluations. We will have the following update rule:

$$\begin{aligned}&\text{pick } i_t \text{ randomly from } \{1,2,...,n\}\\&v_t=g_{i_t}(x_t,u_t)-g_{i_t}(\tilde{x},u_t)+\tilde{g}\\&x_{t+1}=x_t-\eta v_t\end{aligned} \tag{C.2}$$

Note that v_t is an unbiased estimator of the gradient of f_μ meaning that: $\mathbb{E}_{i_t,u_t,\tilde{u}}v_t=\nabla f_\mu(x_t)$. Usually the steps in which we do a full pass over the data and update $\tilde{g}$ are called *outer loop* and the other steps in which we only sample one data point and update x_t are called *inner loop*. We also call m the update frequency of $\tilde{g}$ or the inner loop size. Algorithm 3 describes the procedure of zeroth-order SVRG. As you can see there are multiple ways of updating $\tilde{x}$ at the end of each inner loop. In what follows, we will give an analysis for ZO-SVRG.Our main result is presented in theorem 19.

Theorem 19. *Suppose that we have the objective function defined in* (1.1) *and each f_i is L-smooth and f is α-strongly convex. Further suppose that for optimising f we will use ZO-SVRG introduced in Algorithm 3. If we choose option 2 for updating $\tilde{x}_s$ we will have:*

$$\begin{aligned}\mathbb{E}f(\tilde{x}_{s+1})-f(x^*)\leq\Big(\frac{1}{\alpha\eta(1-16L(d+4)\eta)m}+\frac{16(d+4)L\eta}{(1-16L(d+4)\eta)}\Big)(f_\mu(\tilde{x}_s)-f_\mu(x^*))\\+\frac{12\mu^2L^2(d+4)^3\eta}{2(1-16L(d+4)\eta)}+\frac{16(d+4)\sigma^2\eta}{(1-16L(d+4)\eta)}\end{aligned} \tag{C.3}$$

(The expectation is over the whole randomness of indices i and random directions u that are sampled in the phase s of the algorithm.)

Algorithm 3 Zeroth-Order Stochastic Variance Reduced Gradient (ZO-SVRG)

step size η and update frequency m
initialize $\tilde{x}_0$ randomly
for s *in* $1, 2, ..., T$ **do**
  $\tilde{x} = \tilde{x}_{s-1}$
  $\tilde{g} = \frac{1}{n}\sum_{i=1}^{n} g_i(\tilde{x})$
  $x_1 = \tilde{x}$
  for t *in* $1, 2, ..., m$ **do**
    randomly pick $i_t \in \{1, ..., n\}$
    $x_{t+1} \leftarrow x_t - \eta(g_{i_t}(x_t, u_t) - g_{i_t}(\tilde{x}, u_t) + \tilde{g})$
  end
  option 1: set $\tilde{x}_s = x_m$
  option 2: set $\tilde{x}_s = \frac{1}{m}\sum_{t=1}^{m} x_t$
end
return $\tilde{x}_T$

Assuming that the last two terms in (C.3) are negligible, Theorem 19 shows that if values η and m are chosen such that

$$A = \Big(\frac{1}{\alpha\eta(1 - 16L(d+4)\eta)m} + \frac{16(d+4)L\eta}{(1 - 16L(d+4)\eta)}\Big) < 1 \tag{C.4}$$

then `ZO-SVRG` converges.

Proof. In our proof we used option 2 for updating $\tilde{x}$ following the analysis in theorem 6.5 of Bubeck (2014). We have the standard assumption on the upper bound of the variance: $\|\nabla f_{\mu,i}(x) - \nabla f_\mu(x)\|^2 \leq \sigma^2$. In our analysis we fix a phase $s \geq 1$ of the algorithm where we have $\tilde{x} = \tilde{x}_s$. Assume that x^* is the optimum point of $f_\mu(x)$ and $\tilde{u}$ is the random direction which was picked to compute $\tilde{g}$. We start with the definition of update step in (C.2):

$$\|x_{t+1} - x^*\|^2 = \|x_t - x^*\|^2 - 2\eta v_t^\top (x_t - x^*) + \eta^2 \|v_t\|^2 \tag{C.5}$$

We take $\mathbb{E}_{i_t,u_t,\tilde{u}}$ from the above inequality. Using the fact that v_t is an unbiased estimator of $\nabla f_\mu(x_t)$ we can write:

$$\mathbb{E}_{i_t,u_t,\tilde{u}}\big[v_t^\top (x_t - x^*)\big] = \nabla f_\mu(x_t)^\top (x_t - x^*) \overset{(1.4)}{\geq} f_\mu(x_t) - f_\mu(x^*) \tag{C.6}$$

Next we will upper bound $\mathbb{E}_{i_t,u_t,\tilde{u}} \|v_t\|^2$. Using Lemma 11 and the variance inequality $\mathbb{E}\|X - \mathbb{E}X\|^2 \leq \mathbb{E}\|X\|^2$ we have:

$$\begin{aligned}
\mathbb{E}_{i_t,u_t,\tilde{u}} \|v_t\|^2 &\leq 2\mathbb{E}_{i_t,u_t} \|g_{i_t}(x_t, u_t)\|^2 + 2\mathbb{E}_{i_t,u_t,\tilde{u}} \|g_{i_t}(\tilde{x}, u_t) - \tilde{g}\|^2 \\
&\leq 2\mathbb{E}_{i_t,u_t} \|g_{i_t}(x_t, u_t)\|^2 + 2\mathbb{E}_{i_t,u_t} \|g_{i_t}(\tilde{x}, u_t)\|^2 \\
&\overset{(2.9)}{\leq} 12\mu^2 L^2 (d+4)^3 + 8(d+4)\mathbb{E}_{i_t} \|\nabla f_{\mu,i_t}(x_t)\|^2 + 8(d+4)\mathbb{E}_{i_t} \|\nabla f_{\mu,i_t}(\tilde{x})\|^2
\end{aligned} \tag{C.7}$$

Now we want to upper bound $\mathbb{E}_{i_t} \|\nabla f_{\mu,i_t}(x_t)\|^2$. Using the Remark 15 we have:

$$\mathbb{E}_{i_t} \|\nabla f_{\mu,i_t}(x_t)\|^2 \overset{(A.11)}{\leq} 4L(f_\mu(x_t) - f_\mu(x^*)) + 2\sigma^2 \tag{C.8}$$

Combining (C.6), (C.7) and (C.8) we obtain:

$$\begin{aligned}
\mathbb{E}_{i_t,u_t,\tilde{u}} \|x_{t+1} - x^*\|^2 \leq \|x_t - x^*\|^2 &- 2\eta(1 - 16\eta L(d+4))(f_\mu(x_t) - f_\mu(x^*)) \\
&+ 32(d+4)L\eta^2(f_\mu(\tilde{x}) - f_\mu(x^*)) + 12\mu^2 L^2 (d+4)^3 \eta^2 + 32(d+4)\sigma^2\eta^2
\end{aligned} \tag{C.9}$$

Summing the above inequality over $t = 1, 2, ..., m$ and taking expectation over the whole randomness of $i_1, i_2, ..., i_m$ yields:

$$\begin{aligned}
\mathbb{E}\|x_{m+1} - x^*\|^2 \leq \|x_1 - x^*\|^2 &- 2\eta(1 - 16\eta L(d+4))\mathbb{E}\sum_{t=1}^{m}(f_\mu(x_t) - f_\mu(x^*)) \\
&+ 32m(d+4)L\eta^2(f_\mu(\tilde{x}) - f_\mu(x^*)) + 12m\mu^2 L^2 (d+4)^3 \eta^2 + 32m(d+4)\sigma^2\eta^2
\end{aligned} \tag{C.10}$$

Note that $x_1 = \tilde{x}$ and by strong convexity we have $f_\mu(x) - f_\mu(x^*) \geq \frac{\alpha}{2} \|x - x^*\|^2$. By rearranging the above we can obtain:

$$\begin{aligned}\mathbb{E}f(\frac{1}{m}\sum_{t=1}^{m} x_t) - f(x^*) \leq \Big(\frac{1}{\alpha\eta(1-16L(d+4)\eta)m} + \frac{16(d+4)L\eta}{(1-16L(d+4)\eta)}\Big)(f_\mu(\tilde{x}) - f_\mu(x^*)) \\ + \frac{12\mu^2L^2(d+4)^3\eta}{2(1-16L(d+4)\eta)} + \frac{16(d+4)\sigma^2\eta}{(1-16L(d+4)\eta)}\end{aligned} \tag{C.11}$$

Which is the statement of the proof. As mentioned earlier for ZO-SVRG to converge we should choose η and m such that $A < 1$ in (C.4). Suppose that we want the second term of A to be smaller than $\frac{2}{3}$, then we have:

$$\frac{16(d+4)L\eta}{(1-16L(d+4)\eta)} < \frac{2}{3} \implies \eta < \frac{1}{40(d+4)L} \tag{C.12}$$

Furthermore we want the first term of A in (C.4) to be smaller than $\frac{1}{3}$:

$$\frac{1}{\alpha\eta(1-16L(d+4)\eta)m} < \frac{1}{3} \implies m > \frac{200(d+4)L}{\alpha} \tag{C.13}$$

Hence it is necessary to have $\eta = \mathcal{O}(\frac{1}{dL})$ and $m = \Omega(d\kappa)$ for ZO-SVRG to converge ($\kappa = \frac{L}{\alpha}$). Johnson and Zhang (2013) derive similar values for η and m in their analysis for SVRG. The main difference is that here we have the extra factor of d which is because of using zeroth-order gradient estimates. Plugging in the derived inequalities for η and m in (C.11) we obtain:

$$\mathbb{E}f(\frac{1}{m}\sum_{t=1}^{m} x_t) - f(x^*) \leq A(f_\mu(\tilde{x}) - f_\mu(x^*)) + \frac{\mu^2L(d+4)^2}{4} + \frac{\sigma^2}{L} \tag{C.14}$$

By choosing $\mu < \frac{1}{d\sqrt{L}}$ the second term in (C.14) will vanish eventually, however the term $\frac{\sigma^2}{L}$ is preventing us to get linear convergence rate. If we were able to remove this term then ZO-SVRG would have converged after $\mathcal{O}(\kappa d \log(\frac{1}{\epsilon}))$ iterations.

In order to get rid of the constant term $\frac{\sigma^2}{L}$, we need to reduce σ^2. This could be done by using a zeroth-order oracle with large batch size and population size. Particularly if we use oracle (2.11) from Section 2.3 with batch size b and population size p, every term in the variance bound (2.9) will be divided by $b \times p$. Therefore the choice of values for η and m will also be affected:

$$\eta < \frac{bp}{40(d+4)L}, \qquad\qquad m > \frac{200(d+4)L}{\alpha bp} \tag{C.15}$$

We can conclude this analysis with the following remark:

Remark 20. *In ZO-SVRG algorithm we can choose a large enough batch size and population size such that the $\frac{\sigma^2}{bpL}$ becomes negligible. If we further choose $\mu < \frac{1}{d\sqrt{L}}$, $\eta = \mathcal{O}(\frac{bp}{dL})$ and $m = \Omega(\frac{d\kappa}{bp})$, then ZO-SVRG algorithm will converges to ϵ accuracy of the optimum after $\mathcal{O}(\frac{\kappa d}{bp}\log(\frac{1}{\epsilon}))$ iterations.*
This will be equivalent to $\mathcal{O}((\kappa d + np)\log(\frac{1}{\epsilon}))$ computations of zeroth-order gradients.

The main conclusion of Remark 20 is that ZO-SVRG does not achieve the linear convergence rate of $\mathcal{O}(\log(\frac{1}{\epsilon}))$ with small batch size and population size. Furthermore it shows that with small batch size and population size it can only reach a neighbourhood of the optimum.

Appendix D

Figures

We present the results of the Section 4.5.1 for another data-set. We used the Musk data-set from UCI machine learning repository (Dua and Graff (2017)). This data-set describes a set molecules which are judged by human experts to be musks or non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. It contains $n = 476$ samples and $d = 166$ features. We present the results for non-stochastic and stochastic case in the following figures. For the results of this section we also tuned the update frequency of the population for `(S)GD_ZO_pop_mem` method. However it seems that the results for this method are not very different from Section 4.5.1.

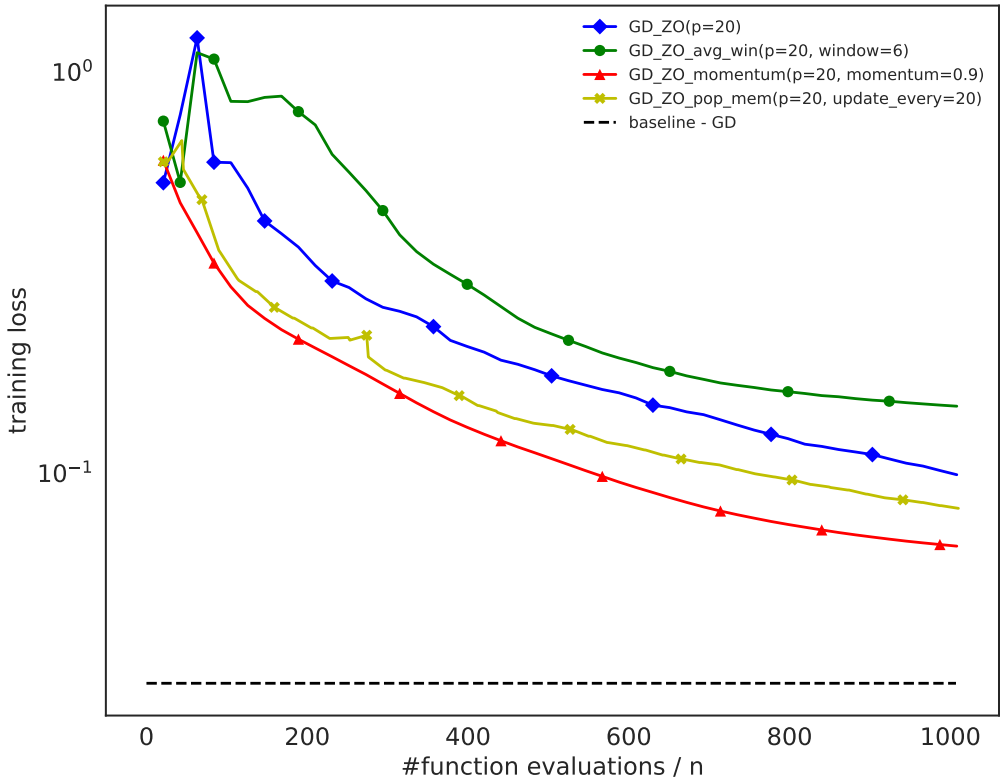
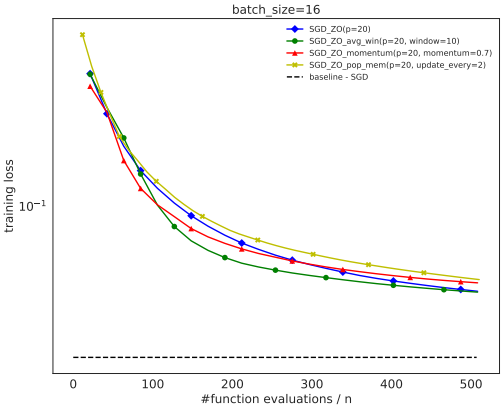
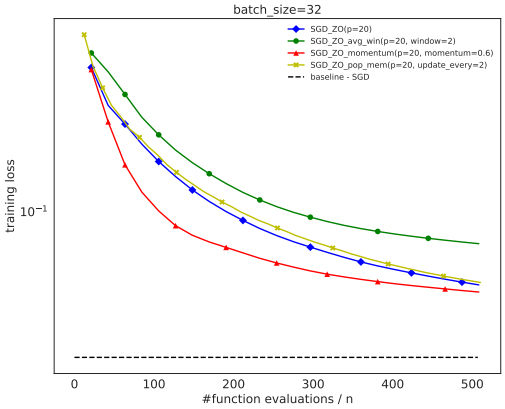


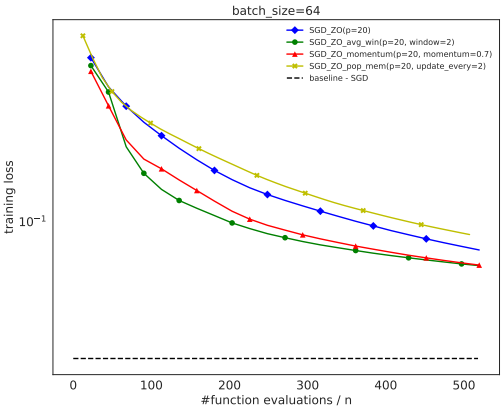
Figure D.1: Comparison of different zeroth-order methods in the non-stochastic case



(a) batch size 16



(b) batch size 32



(c) batch size 64

Figure D.2: Comparison of the proposed methods in the stochastic case

Bibliography

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD. ISBN 978-3-7908-2604-3.

Leon Bottou. *Stochastic Gradient Descent Tricks*, volume 7700 of *Lecture Notes in Computer Science (LNCS)*, pages 430–445. Springer, neural networks, tricks of the trade, reloaded edition, January 2012. URL `https://www.microsoft.com/en-us/research/publication/stochastic-gradient-tricks/`.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Richard Brent. Algorithms for minimization without derivatives. *Englewood Cliffs, Prentice Hall*, 19, 01 2002. doi: 10.2307/2005713.

Sébastien Bubeck. Convex Optimization: Algorithms and Complexity. *arXiv e-prints*, art. arXiv:1405.4980, May 2014.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 15–26, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5202-4. doi: 10.1145/3128572.3140448. URL `http://doi.acm.org/10.1145/3128572.3140448`.

Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009. ISBN 0898716683, 9780898716689.

Jean-Baptiste Cordonnier. Convex optimization using sparsified stochastic gradient descent with memory. page 49 pages, 2018. URL `http://infoscience.epfl.ch/record/262760`.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. *arXiv e-prints*, art. arXiv:1407.0202, Jul 2014.

Juan Du. The frontier of SGD and its variants in machine learning. *Journal of Physics: Conference Series*, 1229:012046, may 2019. doi: 10.1088/1742-6596/1229/1/012046. URL `https://doi.org/10.1088%2F1742-6596%2F1229%2F1%2F012046`.

Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Saeed Ghadimi and Guanghui Lan. Stochastic First- and Zeroth-order Methods for Nonconvex Stochastic Programming. *arXiv e-prints*, art. arXiv:1309.5549, Sep 2013.

Bin Gu, Zhouyuan Huo, and Heng Huang. Zeroth-order Asynchronous Doubly Stochastic Algorithm with Variance Reduction. *arXiv e-prints*, art. arXiv:1612.01425, Dec 2016.

Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *arXiv e-prints*, art. arXiv:1604.00772, Apr 2016.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani,

and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 315–323. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/4937-accelerating-stochastic-gradient-descent-using-predictive-variance-reduction.pdf.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed <today>].

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. A simpler approach to obtaining an O(1/t) convergence rate for the projected stochastic subgradient method. *arXiv e-prints*, art. arXiv:1212.2002, Dec 2012.

Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. Zeroth-order stochastic variance reduction for nonconvex optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3727–3737. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7630-zeroth-order-stochastic-variance-reduction-for-nonconvex-optimization.pdf.

Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtarik. SGD: General Analysis and Improved Rates. *arXiv e-prints*, art. arXiv:1901.09401, Jan 2019.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv e-prints*, art. arXiv:1610.08401, Oct 2016.

Y. E. NESTEROV. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983. URL https://ci.nii.ac.jp/naid/10029946121/en/.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Found. Comput. Math.*, 17(2):527–566, April 2017. ISSN 1615-3375. doi: 10.1007/s10208-015-9296-2. URL https://doi.org/10.1007/s10208-015-9296-2.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12 (1):145 – 151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL http://www.sciencedirect.com/science/article/pii/S0893608098001166.

Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making Gradient Descent Optimal for Strongly Convex Stochastic Optimization. *arXiv e-prints*, art. arXiv:1109.5647, Sep 2011.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv e-prints*, art. arXiv:1609.04747, Sep 2016.

Richard S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Andre Wibisono, Martin J Wainwright, Michael I. Jordan, and John C Duchi. Finite sample convergence rates of zero-order stochastic optimization methods. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1439–1447. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4550-finite-sample-convergence-rates-of-zero-order-stochastic-optimization-methods.

pdf.