Dauphine | PSL
UNIVERSITÉ PARIS

Lecture notes on stochastic gradient methods

Clément W. Royer

M2 IASD & M2 MASH - 2023/2024

- The last version of these notes can be found at:
  https://www.lamsade.dauphine.fr/∼croyer/ensdocs/SG/LectureNotesOML-SG.pdf.
- Comments, typos, etc, can be sent to `clement.royer@lamsade.dauphine.fr`.
- **Major updates of the document**
  - 2023.10.12: First version of the notes.

Foreword

The purpose of these lectures is to introduce the stochastic gradient method. Like any presentation of such a widely studied topic, these notes have their own biases. Those include that of the author as well as the references they are mainly based upon [1, 3, 4]. Rather than giving a necessarily incomplete literature review of all variants of this method, and their applications, these notes intend to convey the main principles behind stochastic gradient. More precisely, the goals that the lectures aim for are the following.

- Understand the motivation behind the stochastic gradient algorithm, and its relevance in a learning setting;
- Provide a comparison between stochastic gradient and gradient descent on both a theoretical and a practical standpoint;
- Review major variants on the stochastic gradient framework;
- Observe the performance of stochastic gradient in practice.

Notations

- Scalars (i.e. reals) are denoted by lowercase letters: $a, b, c, \alpha, \beta, \gamma$.
- Vectors are denoted by **bold** lowercase letters: $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
- Matrices are denoted by **bold** uppercase letters: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.
- Sets are denoted by **bold** uppercase cursive letters : $\mathcal{A}, \mathcal{B}, \mathcal{C}$.
- The set of natural numbers (nonnegative integers) is denoted by $\mathbb{N}$; the set of integers is denoted by $\mathbb{Z}$.
- The set of real numbers is denoted by $\mathbb{R}$. Our notations for the subset of nonnegative real numbers and the set of positive real numbers are $\mathbb{R}_+$ and $\mathbb{R}_{++}$, respectively.
- The notation $\mathbb{R}^d$ is used for the set of vectors with $d \in \mathbb{N}$ real components; although we may not explicitly indicate it in the rest of these notes, we always assume that $d \geq 1$.
- A vector $\boldsymbol{x} \in \mathbb{R}^d$ is thought as a column vector, with $x_i \in \mathbb{R}$ denoting its i-th coordinate in the canonical basis of $\mathbb{R}^d$. We thus write $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$, or, in a compact form, $\boldsymbol{x} = [x_i]_{1 \leq 1 \leq d}$.
- Given a column vector $\boldsymbol{x} \in \mathbb{R}^d$, the corresponding row vector is denoted by $\boldsymbol{x}^{\mathrm{T}}$, so that $\boldsymbol{x}^{\mathrm{T}} = [x_1 \ \cdots \ x_d]$ and $[\boldsymbol{x}^{\mathrm{T}}]^{\mathrm{T}} = \boldsymbol{x}$. The scalar product between two vectors in $\mathbb{R}^d$ is defined as $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^d x_i y_i$.
- The Euclidean norm of a vector $\boldsymbol{x} \in \mathbb{R}^d$ is defined by $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$.
- We use $\mathbb{R}^{n \times d}$ to denote the set of real rectangular matrices with n rows and d columns, where n and d will always be assumed to be at least 1. If $n = d$, $\mathbb{R}^{d \times d}$ refers to the set of square matrices of size d.
- We identify a matrix in $\mathbb{R}^{d \times 1}$ with its corresponding column vector in $\mathbb{R}^d$.
- Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$, $\boldsymbol{A}_{ij}$ refers to the coefficient from the i-th row and the j-th column of $\boldsymbol{A}$: the diagonal of $\boldsymbol{A}$ is given by the coefficients $\boldsymbol{A}_{ii}$. Provided this notation is not ambiguous, we use the notations $\boldsymbol{A}$, $[\boldsymbol{A}_{ij}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq d}}$ and $[\boldsymbol{A}_{ij}]$ interchangeably.
- For every $d \geq 1$, $\mathbf{I}_d$ refers to the identity matrix in $\mathbb{R}^{d \times d}$ (with 1s on the diagonal and 0s elsewhere).

Contents

Chapter 1

Introduction

This course is concerned with optimization problems arising in data-related applications. Such formulations have gained tremendous interest in recent years, due to the increase in computational power that enable significant advances in fields such as image processing. One of the most fundamental tools behind data science is optimization,that combines mathematical formulations and algorithmic procedures. We describe below the motivation behind studying optimization techniques tailored to data-related applications, as well as the characteristics of the associated problems.

1.1 Motivation

The words *machine learning* are widely used as a way to characterize any task that involves manipulating data : nevertheless, their precise meaning can be difficult to formalize, as other keywords such as *data mining*, *data analysis*, *artificial intelligence* or *Big Data* also denote fields that involve data and/or a learning process. In these notes, we focus on the link between data-related tasks and optimization; although we will denote our applications of interest as pertaining to machine learning, we point out that a more general, possibly better suited categorization would be that of **data science**. For the purpose of these lectures, we will indeed consider machine learning through two main goals:

1) Extract patterns from data, possibly in terms of statistical properties;

2) Use this information to infer or make predictions about yet unseen data.

A number of such machine learning tasks involve an optimization component. As a result, for the purpose of these notes, we will view machine learning as a field making use of statistics and optimization, with the latter being our area of interest. Nevertheless, we point out that computer science features such as data management and parallel computing have also been instrumental to the success of machine learning, and thus should eventually be integrated with optimization to form efficient algorithms.

1.2 General formulation

1.2.1 From expected to empirical risk

We consider an input space $\mathcal{A}$ and an output space $\mathcal{Y}$. Our goal is to determine a mapping $h : \mathcal{A} \to \mathcal{Y}$ such that, for every input $\boldsymbol{a} \in \mathcal{A}$, the value $h(\boldsymbol{a})$ is an accurate prediction of the true output $\boldsymbol{y} \in \mathcal{Y}$.

Suppose that the examples in our dataset are sampled from a joint distribution $p(\boldsymbol{a}, \boldsymbol{y})$. We seek a predictor function h that yields a small expected risk, where

$$R(h) := \mathbb{P}\left(h(\boldsymbol{a}) \neq \boldsymbol{y}\right) = \mathbb{E}\left[\mathbf{1}(h(\boldsymbol{a}) \neq \boldsymbol{y})\right], \tag{1.2.1}$$

where $\mathbf{1}(\cdot)$ denotes the indicator function of an event. In practice, we rarely know the distribution of the data, and we can only access a sample $\{(\boldsymbol{a}_i, \boldsymbol{y}_i)\}_{i=1}^n$ of the distribution. In this case, we can quantify how good our prediction is *on this dataset* by considering the **empirical risk** function defined as

$$R_n(h) := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(h(\boldsymbol{a}_i) \neq \boldsymbol{y}_i). \tag{1.2.2}$$

Unlike the expected risk function, the empirical risk function can usually be computed, as it corresponds to data points that are available. Through arguments based on the law of large numbers, one can ensure that, with sufficiently many samples, the difference between empirical and expected risk can be bounded with high probability.

1.2.2 Loss and prediction functions

The previous measures of risk are exact, in that they directly measure whether a model correctly predicts an output given the input. Their definition can however lead to functions of h that are discontinuous or combinatorial in nature. This can pose numerous challenges in designing algorithms that compute the model with the lowest risk possible. For this reason, a common practice consists in introducing a **loss function**, that quantifies the discrepancy (or lack thereof) between the output of a model $h(\boldsymbol{a})$ and the true output $\boldsymbol{y}$. That is, for every sample $(\boldsymbol{a}, \boldsymbol{y})$ from the desired distribution, we use

$$\mathbf{1}(h(\boldsymbol{a}) \neq \boldsymbol{y}) \approx \ell(h(\boldsymbol{a}), \boldsymbol{y}),$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathcal{Y}$ is a given loss function.

In addition, rather than considering a generic set of models, we assume that models can be characterized by means of a vector in $\mathbb{R}^d$ (where d can be extremely large). Therefore, we will use the notation $h(\cdot; \boldsymbol{x}) : \boldsymbol{a} \mapsto h(\boldsymbol{a}; \boldsymbol{x})$.

Overall, we will approximate the expected risk as follows:

$$R(h) \approx R(\boldsymbol{x}) := \int_{\mathcal{A}\times\mathcal{Y}} \ell(h(\boldsymbol{a}; \boldsymbol{x}); \boldsymbol{y})) = \mathbb{E}\left[\ell(h(\boldsymbol{a}; \boldsymbol{x}); \boldsymbol{y})\right], \tag{1.2.3}$$

while the empirical risk will be estimated by

$$R_n(h) \approx R_n(\boldsymbol{x}) := \frac{1}{n}\sum_{i=1}^{n} \ell(h(\boldsymbol{a}_i; \boldsymbol{x}); \boldsymbol{y}_i)). \tag{1.2.4}$$

Our learning goal will then be to compute a model (that is, a vector $\boldsymbol{x}$) that yields the lowest value of the empirical risk. As a result, we consider the following **empirical risk minimization** (ERM) problem :

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n} \ell(h(\boldsymbol{a}_i; \boldsymbol{x}); \boldsymbol{y}_i)). \tag{1.2.5}$$

In the next section, we will see some examples of such ERM problems.

Remark 1.2.1 *Although computing a model based on the empirical risk represents a reasonable approach to computing a good model, the ideal goal would be to compute a solution that would* **generalize** *to unseen examples from the distribution. This is a very challenging issue from an optimization perspective, as optimization classically assumes that the problem formulation encapsulates all there is to know about this problem.*

1.3 Examples of learning models

1.3.1 Linear regression

Linear least squares is arguably the most classical problem in data analysis. We consider a dataset $\{(\boldsymbol{a}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{a}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Our goal is to compute a linear model that best fits (or explains) the data. We define this model as a function $h : \mathbb{R}^d \to \mathbb{R}$, and we parameterize it through a vector $\boldsymbol{x} \in \mathbb{R}^d$, so that for any $\boldsymbol{a} \in \mathbb{R}^d$, we have $h(\boldsymbol{a}) = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}$. For every example $(\boldsymbol{a}_i, y_i)$ in the dataset, we evaluate how we fit the data based on the squared error $(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} - y_i)^2$. We then compute a model by solving the following optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{2n}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2. = \frac{1}{n}\sum_{i=1}^{n} \tfrac{1}{2}\left[(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} - y_i)^2 + \lambda\|\boldsymbol{x}\|^2\right], \tag{1.3.1}$$

where $\lambda > 0$ is a regularization parameter. From an optimizer's point of view, problem (1.3.1) is well understood: this is a strongly convex, quadratic problem, and its solution can be computed in close form.

In a typical linear regression setting, one assumes that there exists an underlying truth but that the measurements are noisy, i.e.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \in \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a vector with i.i.d. entries following a standard normal distribution: this is illustrated in Figure 1.1.

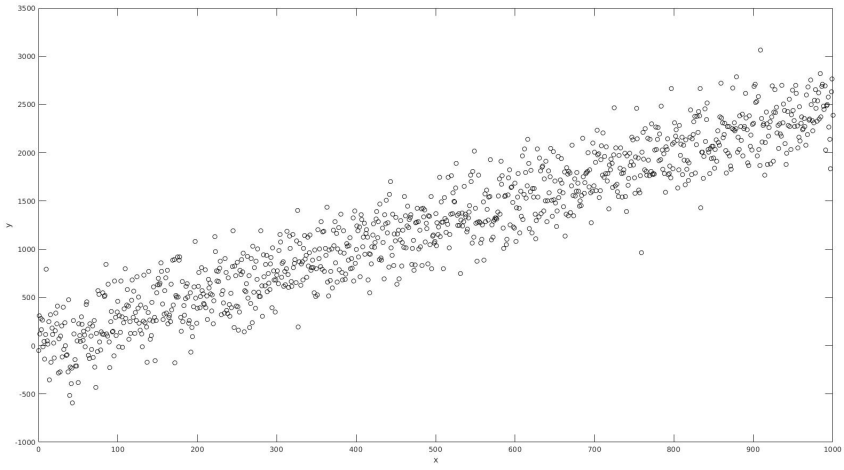

Figure 1.1: Noisy data generated from a linear model with Gaussian noise.

In this setting, we wish to compute the most likely value for $\boldsymbol{x}^*$, while being robust to variance in the data. To this end, we suppose that $\boldsymbol{y}$ follows a Gaussian distribution of mean $\boldsymbol{A}\boldsymbol{x}$ and of covariance matrix $\mathbf{I}$. We also assume a prior Gaussian distribution on the entries of $\boldsymbol{x}$, in order to reduce the variance with respect to the data. As a result, an estimate of $\boldsymbol{x}^*$, called the maximum a posteriori estimator, can be computed by solving

$$\max_{\boldsymbol{x}\in\mathbb{R}^d} L(y_1,\ldots,y_n;\boldsymbol{x}) := \left[\frac{1}{\sqrt{2\pi}}\right]^m \exp\left(-\frac{1}{2}\sum_{i=1}^m(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}-y_i)^2 - \frac{\lambda}{2}\|\boldsymbol{x}\|^2\right). \tag{1.3.2}$$

The solutions of this maximization problem are the same than the solutions of the linear least-squares problem (1.3.1). The resulting solution can be shown to possess very favorable statistical properties: in particular, for λ close to 0, its expected value is close to $\boldsymbol{x}^*$.

Linear regression (with or without regularization) has been extensively studied in optimization and statistics; however, when the number of samples is extremely large, it still poses a number of challenges in practice, as the solution of the problem cannot be computed exactly.

1.3.2 Neural networks

Neural networks have enabled the most impressive, recent advances in perceptual tasks such as image recognition and classification. Thanks to the increase in computational capabilities over the past decade, it is now possible to train extremely deep and wide neural networks, so that they can learn efficient representations of the data.

Given an input vector $\boldsymbol{a}_i \in \mathbb{R}^{d_0}$, a neural network represents a prediction function $h : \mathbb{R}^{d_0} \to \mathbb{R}^{d_J}$, which applies a series of transformations in layers $\boldsymbol{a}_i = \boldsymbol{a}_i^{(0)} \mapsto \boldsymbol{a}_i^{(1)} \mapsto \cdots \mapsto \boldsymbol{a}_i^{(J-1)} \mapsto \boldsymbol{a}_i^{(J)}$. The j-th layer typically performs the following transformation:

$$\boldsymbol{a}_i^{(j)} \;=\; \boldsymbol{\sigma}\left(\boldsymbol{W}_j\boldsymbol{a}_i^{(j-1)} + \boldsymbol{b}_j\right) \in \mathbb{R}^{d_j}, \tag{1.3.3}$$

where $\boldsymbol{W}_j \in \mathbb{R}^{d_j\times d_{j-1}}$, $\boldsymbol{b}_j \in \mathbb{R}^{d_j}$ and $\boldsymbol{\sigma} : \mathbb{R}^{d_j} \to \mathbb{R}^{d_j}$ is a componentwise nonlinear function, e.g. $\boldsymbol{\sigma}(\boldsymbol{y}) = \left[\frac{1}{1+\exp(-y_i)}\right]_i$ (sigmoid function) or $\boldsymbol{\sigma}(\boldsymbol{y}) = [\max(0,y_i)]_i$. As a result, we have $\boldsymbol{a}_i^{(J)} = h(\boldsymbol{a}_i;\boldsymbol{x})$, where $\boldsymbol{x}\in\mathbb{R}^d$ gathers all the parameters $\{(\boldsymbol{W}_1,\boldsymbol{b}_1),\ldots,(\boldsymbol{W}_J,\boldsymbol{b}_J)\}$ of the layers.

The optimization problem corresponding to training this neural network architecture involves a training set $\{(\boldsymbol{a}_i,y_i)\}_{i=1}^n$ and the choice of a loss function ℓ. It usually results in the following formulation

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^n \ell\left(h(\boldsymbol{a}_i;\boldsymbol{x}),y_i\right). \tag{1.3.4}$$

This optimization problem is highly nonlinear and nonconvex in nature, which makes it particularly difficult to solve using algorithms such as gradient descent. Moreover, it typically involves costly algebraic operations, as the number of layers and/or parameters is tremendously large in modern deep neural network architectures. Therefore, problem (1.3.4) also possesses characteristics that are not accounted for in its formulation. The optimization algorithms that efficiently tackle this problem are those that can both guarantee convergence and perform well in practice.

Chapter 2

Stochastic gradient methods

In this chapter, we describe the stochastic gradient method in the context of finite-sum problems, which we introduce in the next section. We will then motivate the use of stochastic gradient methods in this setting, then discuss a basic framework from an algorithmic and theoretical point of view.

2.1 Finite-sum optimization and gradient descent

Suppose that we have access to data samples $\{(\boldsymbol{a}_i, y_i)\}_{i=1}^n$, $\boldsymbol{a}_i \in \mathbb{R}^{d_a}$, $y_i \in \mathbb{R}$, that are drawn from an unknown distribution. As in the examples described in the previous chapter, we seek a predictor function or a model h such that $h(\boldsymbol{a}_i) \approx y_i$ for every $i = 1, \ldots, n$. Rather than optimizing over a space of models, we assume that a given model is defined by means of a vector $\boldsymbol{x} \in \mathbb{R}^d$ (i.e. $h(\boldsymbol{a}_i) = h(\boldsymbol{a}_i; \boldsymbol{x})$). Therefore, we only need to determine the vector $\boldsymbol{x}$ in order to obtain the model.

To assess the accuracy of our model in predicting the data, we make use of a loss function $\ell : (h, y) \mapsto \ell(h, y)$, that penalizes pairs (h, y) for which $h \neq y$. The loss at a given sample of the dataset thus is $\ell(h(\boldsymbol{a}_i; \boldsymbol{x}), y_i)$: in order to account for all samples, we consider the average of all losses as our objective to be minimized. This gives rise to the following optimization problem.

Definition 2.1.1 (Finite-sum optimization problem) *Let $\{(\boldsymbol{a}_i, y_i)\}_{i=1}^n$ be a dataset where for every $i = 1, \ldots, n$, $\boldsymbol{a}_i \in \mathbb{R}^{d_a}$ and $y_i \in \mathbb{R}$, a class of predictor functions $\{h(\cdot; \boldsymbol{x})\}_{\boldsymbol{x} \in \mathbb{R}^d}$ and a loss function ℓ, we define the corresponding optimization problem:*

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{x}), \qquad f_i(\boldsymbol{x}) := \ell(h(\boldsymbol{a}_i; \boldsymbol{x}), y_i)\ \forall i = 1, \ldots, n. \tag{2.1.1}$$

One key property of the formulation (2.1.1) is that every term in the finite sum only involves **one example from the dataset**.

Solving the problem with gradient descent Suppose that the functions f_i are continously differentiable. In that case, the objective function f in (2.1.1) also is continuously differentiable, and we can apply the gradient descent method.[1] The k-th iteration of this method is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - \frac{\alpha_k}{n} \sum_{i=1}^n \ell(h(\boldsymbol{x}; \boldsymbol{a}_i), y_i),$$

[1] See the notes from the first three sessions by Gabriel Peyré for more details about gradient descent.

where $\alpha_k > 0$ is a given stepsize. From this update, we see that one iteration of gradient descent requires to go over the **entire dataset** in order to compute the gradient vector. In a big data setting where the number of samples n is very large, this cost can be prohibitive.

Remark 2.1.1 *In stochastic optimization, the data samples might be generated directly from the distribution, and be available in a streaming fashion. Instead of involving a discrete average on the sample, the resulting optimization problem would involve a mathematical expectation of the form*

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} \mathbb{E}_{(\boldsymbol{a},y)} \left[f_{(\boldsymbol{a},y)}(\boldsymbol{x}) \right].$$

In such a context, the full gradient may not be computable, even if the underlying function is smooth. However, most of the reasoning in the next sections will apply to this setting.

2.2 Stochastic gradient framework

2.2.1 Algorithm

At its core, the idea of the stochastic gradient method is remarkably simple. Starting from the problem $\min_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^n f_i(\boldsymbol{x})$, and assuming each component function f_i is differentiable, the method picks an index i at random and takes a step in the direction of the negative gradient of the component function f_i.

Algorithm 1: Stochastic gradient method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**
  Compute a stepsize or learning rate $\alpha_k > 0$.
  Draw a random index $i_k \in \{1, \ldots, n\}$.
  Compute the new iterate as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{x}_k). \tag{2.2.1}$$

end

The key motivation for this process is that using a single data point at a time results in updates that are n **times cheaper than a full gradient step**. Note, however, that using a single component does not necessarily lead to convergence, as illustrated by the following example.

Example 2.2.1 *Consider the problem minimize$_{x\in\mathbb{R}} \frac{1}{2}(f_1(x) + f_2(x))$ with $f_1(x) = 2x^2$ and $f_2 = -x^2$. Starting from $x_k > 0$, drawing $i_k = 2$ will necessarily lead to an increase in the function value.*

In finite-sum problems arising from machine learning, the data samples are correlated enough that an update according to one sample might lead to improvement with respect to other samples as well: this is a key reason for the success of stochastic gradient methods in this setting.

Remark 2.2.1 *Algorithm 1 is often referred to as Stochastic Gradient Descent, or SGD, by analogy with Gradient Descent. However, this algorithm is not a descent method in general (as we will see in the next section, it can however produce descent in expectation). For this reason, we will refer to these methods as stochastic gradient algorithms.*

2.2.2 Batch stochastic gradient methods

The main part of Algorithm 1 consists in the update

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{x}_k),$$

where the index i_k is drawn at random.. One canalso consider stochastic gradient estimates that are built using *several* samples at once : this is the idea behind batch stochastic gradient.

Formally, the update of a batch stochastic gradient method is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(\boldsymbol{x}_k) \tag{2.2.2}$$

where $S_k \subset \{1, \ldots, n\}$ is drawn at random. When S_k consists in a single index, we recover the usual stochastic gradient algorithm; if $|S_k| = n$ and the n indices are drawn without replacement, then $S_k = \{1, \ldots, n\}$, and we recover the usual gradient descent algorithm[2].

Overall, two batch regimes can be distinguished:

- $|S_k| \approx n$, which has a cost essentially equivalent to that of a full gradient update;
- $|S_k| = n_b \ll n$, also called mini-batching, which may be advantageous in theory and variance reduction while still being affordable in practice. The resulting method is called mini-batch SG.

We note that it is possible to provide a unified view of batch and stochastic gradient methods; of more interest to us is the comparison of the performance of these various schemes, which is usually done in terms of **epochs**.

Definition 2.2.1 (Epoch) *For problem* (2.1.1)*, an epoch represents n calculations of a sample gradient ∇f_i.*

As a result, one iteration of gradient descent is an epoch, but an epoch corresponds to n iterations of Algorithm 1 and n/n_b iterations of a batch stochastic gradient method with a fixed batch size of n_b.

2.3 Theoretical analysis of stochastic gradient

In this section, we establish convergence guarantees for the stochastic gradient method, in the strongly convex and nonconvex settings. We specify the assumptions under which our analysis is performed, then discuss the dependency of the convergence results to the choice of the stepsize.

2.3.1 Assumptions and first properties

We now describe the main arguments in deriving convergence rates for stochastic gradient. For simplicity, and ease of exposure, we will focus on a specific class of functions.

[2] The gradient descent method is sometimes called the batch gradient algorithm in machine learning.

Assumption 2.3.1 *The objective function $f = \frac{1}{n}\sum_{i=1}^n f_i$ belongs to $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ for $L > 0$, i.e. f is continuously differentiable, and its gradient is L-Lipschitz continuous:*

$$\forall (\boldsymbol{x}, \boldsymbol{w}) \in (\mathbb{R}^d)^2, \qquad \|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{w})\| \leq L\|\boldsymbol{x} - \boldsymbol{w}\|.$$

In addition, there exists $f_{low} \in \mathbb{R}$ such that for every $\boldsymbol{x} \in \mathbb{R}^d$, $f(\boldsymbol{x}) \geq f_{low}$. Moreover, every function f_i belongs to $\mathcal{C}^1(\mathbb{R}^d)$.

The smoothness assumption above is instrumental to analyzing optimization schemes, as it provides the following upper bound on the objective:

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2. \tag{2.3.1}$$

For the gradient descent method, one can leverage this inequality to guarantee descent at iteration k (that is, $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$) for an appropriate choice of stepsize

In the case of Algorithm 1, it is not meaningful to study $f(\boldsymbol{x}_{k+1})$, as this value is random. We can however look at its expected value over i_k, which leads to the following result.

Proposition 2.3.1 *Under Assumption 2.3.1, consider the k-th iteration of Algorithm 1. Then,*

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1})\right] \leq f(\boldsymbol{x}_k) - \alpha_k \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}\, \mathbb{E}_{i_k}\left[\nabla f_{i_k}(\boldsymbol{x}_k)\right] + \frac{L\alpha_k^2}{2}\, \mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{x}_k)\|^2\right].$$

In light of Example 2.2.1, we know that the stochastic gradient method may not lead to decrease in the function value. However, we will provide guarantees in expectation under additional assumptions on how the stochastic gradient estimate $\nabla f_{i_k}(\boldsymbol{x}_k)$.

Assumption 2.3.2 (Assumptions on stochastic gradient) *At any iteration of Algorithm 1 of index k, i_k is drawn such that:*

1. *The index i_k does not depend from the previous indices $i_0, \ldots, i_{k-1}$;*
2. $\mathbb{E}_{i_k}\left[\nabla f_{i_k}(\boldsymbol{x}_k)\right] = \nabla f(\boldsymbol{x}_k)$;
3. $\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{x}_k)\|^2\right] \leq \sigma^2 + \|\nabla f(\boldsymbol{x}_k)\|^2$ *with* $\sigma^2 \geq 0$.

The second property of Assumption 2.3.2 forces the stochastic gradient $\nabla f_{i_k}(\boldsymbol{x}_k)$ to be an unbiased estimate of the true gradient $\nabla f(\boldsymbol{x}_k)$. The third property controls the variance of the norm of this stochastic gradient, so as to control the variations in its magnitude due to noise. Note that the term in $\|\nabla f(\boldsymbol{x}_k)\|^2$ could have been omitted, and is kept to highlight similarities with the gradient descent analysis later on.

Several strategies can be designed to draw an index i_k that satisfies these properties, the most classical of which is given below.

Example 2.3.1 (Uniform sampling) *Suppose that the indices $\{i_k\}_k$ are i.i.d. random variables that are uniformly drawn at random in $\{1, \ldots, n\}$. Then Algorithm 1 satisfies the first two properties of Assumption 2.3.2.*

The third property in Assumption 2.3.2 may require additional knowledge on the problem[3]. For instance, if there exists $M > 0$ such that $\|\nabla f_{i_k}(\boldsymbol{x}_k)\| \leq M$ for all k (which is the case if the iterates remain in a compact set), the property will hold.

Under these assumptions together with Proposition 2.3.1, we obtain the following result.

Proposition 2.3.2 *Under Assumptions 2.3.1 and 2.3.2, at the k-th iteration of Algorithm 1, one has*

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1})\right] \leq f(\boldsymbol{x}_k) - \left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{L\alpha_k^2}{2}\sigma^2. \tag{2.3.2}$$

A stochastic gradient update will thus lead to decrease **in expectation**. Such a property suffices to derive convergence rates (or complexity results) for stochastic gradient applied to strongly convex, convex or nonconvex problems. Those results heavily depend upon the formula for the step sizes $\{\alpha_k\}_k$.

2.3.2 Analysis in the strongly convex case

Most of our analysis will focus on the strongly convex setting. This will allow us to highlight the main properties of stochastic gradient techniques. For this section, we will thus operate under the following assumption.

Assumption 2.3.3 *There exists $\mu > 0$ such that the objective function is μ-strongly convex, i.e. for every $(\boldsymbol{x}, \boldsymbol{w}) \in (\mathbb{R}^d)^2$, we have*

$$f(\boldsymbol{w}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{w} - \boldsymbol{x}\|^2. \tag{2.3.3}$$

and possesses a unique global minimizer $\boldsymbol{x}^*$. We let $f^* = f(\boldsymbol{x}^*)$.

When the function also satisfies the Taylor bound (2.3.1), we have $L \geq \mu$. Assumption 2.3.3 has the following useful consequence.

Lemma 2.3.1 *Let Assumptions 2.3.1 and 2.3.3 hold. Then, for every $\boldsymbol{x} \in \mathbb{R}^d$, we have*

$$\|\nabla f(\boldsymbol{x})\|^2 \;\geq\; 2\mu\left(f(\boldsymbol{x}) - f^*\right). \tag{2.3.4}$$

Proof. Consider the characterization of strong convexity (2.3.3); for any points $(\boldsymbol{x}, \boldsymbol{w}) \in (R^n)^2$, we have

$$f(\boldsymbol{w}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{w} - \boldsymbol{x}\|^2.$$

Minimizing both sides with respect to $\boldsymbol{w}$ lead to $\boldsymbol{w} = \boldsymbol{x}^*$ on the left-hand side, and $\boldsymbol{w} = \boldsymbol{x} - \frac{1}{\mu}\nabla f(\boldsymbol{x})$ on the right-hand side[4]. As a result, we obtain

$$\begin{aligned} f^* &\geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}\left[-\frac{1}{\mu}\nabla f(\boldsymbol{x})\right] + \frac{\mu}{2}\|-\frac{1}{\mu}\nabla f(\boldsymbol{x})\|^2 \\ f^* &\geq f(\boldsymbol{x}) - \frac{1}{2\mu}\|\nabla f(\boldsymbol{x})\|^2. \end{aligned}$$

[3] Or a more general assumption that is out of the scope of these lectures.

[4] The right-hand side is a simple convex quadratic function of $\boldsymbol{w}$, and its first-order optimality condition reads $\nabla f(\boldsymbol{x}) + \mu(\boldsymbol{w} - \boldsymbol{x}) = \boldsymbol{0}$.

By re-arranging the terms, we arrive at the desired result. □

Our results will be provided in expectation. For any k, we will exploit the independence assumption on the indices i_k and write

$$\mathbb{E}\left[f(\boldsymbol{x}_k)\right] = \mathbb{E}_{i_0}\left[\mathbb{E}_{i_1}\left[\ldots \mathbb{E}_{i_{k-1}}\left[f(\boldsymbol{x}_k)\right]\right]\right].$$

Note that this quantity will be deterministic (fixed) with respect to every random index i_j with $j \geq k$. For this reason, we may also write $\mathbb{E}\left[f(\boldsymbol{x}_k)\right]$ to denote the expected value over all indices $i_0, \ldots, i_j$.

We first provide a global rate result in the case of a constant step size.

Theorem 2.3.1 (SG with constant stepsize) *Let Assumptions 2.3.1, 2.3.2 and 2.3.3 hold. Consider Algorithm 1 applied with a constant stepsize*

$$\alpha_k = \alpha \in \left(0, \tfrac{1}{L}\right] \forall k.$$

Then, for any $K \geq 1$, we have

$$\mathbb{E}\left[f(\boldsymbol{x}_K) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu} + (1 - \alpha\mu)^K \left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu}\right]. \tag{2.3.5}$$

Proof. Consider the k-th iteration for $k \in \{0, \ldots, K-1\}$. Applying the result of Proposition (2.3.2), we have

$$\begin{aligned}
\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] &\leq -\left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{L\alpha_k^2}{2}\sigma^2 \\
&\leq -2\mu\left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\left(f(\boldsymbol{x}_k) - f^*\right) + \frac{L\alpha_k^2}{2}\sigma^2 \\
&= -2\mu\alpha\left(1 - \frac{L\alpha}{2}\right)\left(f(\boldsymbol{x}_k) - f^*\right) + \frac{L\alpha}{2}\sigma^2.
\end{aligned}$$

Using that $\alpha \leq \frac{1}{L}$ then gives

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \leq -\mu\alpha(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2}\sigma^2.$$

Noticing that $\mathbb{E}_{i_k}\left[f^* - f(\boldsymbol{x}_k)\right] = f^* - f(\boldsymbol{x}_k)$, the left-hand side can be modified by adding and subtracting f^*, leading to

$$\begin{aligned}
\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] + f^* - f(\boldsymbol{x}_k) &\leq -\mu\alpha(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2}\sigma^2 \\
\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] &\leq (1 - \mu\alpha)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2}\sigma^2.
\end{aligned}$$

Note that $\frac{1}{L} \leq \frac{1}{\mu}$, thus $1 - \mu\alpha \in (0, 1)$. One final subtraction on both sides gives

$$\begin{aligned}
\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] - \frac{L\alpha}{2\mu}\sigma^2 &\leq (1 - \mu\alpha)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2}\sigma^2 - \frac{L\alpha}{2\mu}\sigma^2 \\
&= (1 - \mu\alpha)\left[(f(\boldsymbol{x}_k) - f^*) - \frac{L\alpha}{2\mu}\sigma^2\right].
\end{aligned}$$

Finally, taking the expected value with respect to every index $i_0, \ldots, i_k$, we arrive at

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] - \frac{L\alpha}{2\mu}\sigma^2 \;\leq\; (1-\mu\alpha)\left[\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] - \frac{L\alpha}{2\mu}\sigma^2\right].$$

By applying this inequality recursively for $k = K-1, \ldots, 0$, we arrive at the desired result. □

At first glance, the result of Theorem 2.3.1 suggests that the convergence rate of stochastic gradient (in expectation) is similar to that of gradient descent. Indeed, for gradient descent, one can show under the same assumption on α_k that

$$f(\boldsymbol{x}_K) - f^* \leq (1-\alpha\mu)^K \left[f(\boldsymbol{x}_0) - f^*\right].$$

In the case of Algorithm 1, however, we observe that residual terms appear in the convergence rate, that are related to the variance of the gradient estimator. As a result, SG with constant stepsize can only be guaranteed to converge towards a **neighborhood** of the optimal function value f^*. The result of Theorem 2.3.1 illustrates this interplay between the choice of the (constant) stepsize and the residual noise in the problem: if we set α to a small value, the variance-related terms are reduced, but the convergence rate of the method is slower.

Remark 2.3.1 *It is possible to use Markov's inequality (see Appendix A.5) to obtain results in probability rather than in expected value.*

A practical constant stepsize approach A common practical strategy in machine learning consists in running the algorithm with a value α until the method stalls (which can indicate that the smallest neighborhood attainable with this stepsize choice has been reached). When that occurs, the stepsize can be reduced, and the algorithmic run can continue until it stalls again, then the stepsize will be further reduced, etc (say $\alpha, \alpha/2, \alpha/4$, etc). This process can lead to convergence guarantees, in that it is possible to reach any neighborhood of the optimal value f^*. However, the convergence rate is sublinear, in the sense that

$$\mathbb{E}\left[f(\boldsymbol{x}_K) - f^*\right] \leq \mathcal{O}\left(\frac{1}{K}\right)$$

This choice of stepsize is adaptive, in that it is designed to reach closer and closer neighborhoods as the algorithm proceeds. However, it requires the method to be able to detect stalling, and act upon it.

In the original stochastic gradient method (proposed by Robbins and Monro in 1951), the stepsize sequence was required to satisfy

$$\sum_{k=0}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty,$$

which implies that $\alpha_k \to 0$. In our next result, we thus consider the case of diminishing stepsizes.

Theorem 2.3.2 (SG with diminishing stepsize) *Let Assumptions 2.3.1, 2.3.2 and 2.3.3, and consider Algorithm 1 applied with a decreasing stepsize sequence $\{\alpha_k\}_k$ satisfying*

$$\alpha_k = \frac{\beta}{k+\gamma},$$

where $\beta > \frac{1}{\mu}$ *and* $\gamma > 0$ *is chosen such that* $\alpha_0 = \frac{\beta}{\gamma} \leq \frac{1}{L}$. *Then, for any* $K \geq 1$,

$$\mathbb{E}\left[f(\boldsymbol{x}_K) - f^*\right] \leq \frac{\nu}{\gamma + K}, \tag{2.3.6}$$

where

$$\nu = \max\left\{\gamma(f(\boldsymbol{x}_0) - f^*), \frac{\beta^2 L\sigma^2}{2(\beta\mu - 1)}\right\}.$$

Proof. We proceed as in the proof of Theorem 2.3.1, but invoking the result of Proposition 2.3.1. Namely, for any $k \in \{0, \ldots, K-1\}$, we have

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \leq -\left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{L\alpha_k^2}{2}\sigma^2.$$

For every k, we have $1 - \frac{L\alpha_k}{2} \geq 1 - \frac{L\alpha_0}{2} \geq \frac{1}{2}$. Therefore,

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] &\leq -\frac{1}{2}\alpha_k\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{L\alpha_k^2}{2}\sigma^2 \\ &\leq -\alpha_k\mu(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha_k^2}{2}\sigma^2.\end{aligned}$$

By introducing f^* on the left-hand side and taking the expectation over all indices $i_0, \ldots, i_k$, we obtain :

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] \leq (1 - \alpha_k\mu)\,\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] + \frac{L\alpha_k^2}{2}\sigma^2. \tag{2.3.7}$$

We now prove the desired result (2.3.6) by induction. The result is clearly true for $k = 0$, since

$$\mathbb{E}\left[f(\boldsymbol{x}_0) - f^*\right] = \frac{\gamma}{\gamma + 0}(f(\boldsymbol{x}_0) - f^*) \leq \frac{\nu}{\gamma + 0}.$$

Suppose now that (2.3.6) holds at iteration k. Then, by (2.3.7), we obtain

$$\begin{aligned}\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] &\leq (1 - \alpha_k\mu)\,\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] + \frac{L\alpha_k^2}{2}\sigma^2 \\ &\leq (1 - \alpha_k\mu)\frac{\nu}{\gamma + k} + \frac{L\alpha_k^2}{2}\sigma^2 \\ &= \left(1 - \frac{\mu\beta}{\gamma + k}\right)\frac{\nu}{\gamma + k} + \frac{1}{2}\frac{\beta^2 L\sigma^2}{(\gamma + k)^2} \\ &= \frac{\gamma + k - \mu\beta}{(\gamma + k)^2}\nu + \frac{1}{2}\frac{\beta^2 L\sigma^2}{(\gamma + k)^2} \\ &= \frac{\gamma + k - 1}{(\gamma + k)^2}\nu - \frac{\mu\beta - 1}{(\gamma + k)^2}\nu + \frac{1}{2}\frac{\beta^2 L\sigma^2}{(\gamma + k)^2}.\end{aligned}$$

Using the relation

$$(1 - \mu\beta)\nu + \frac{\beta^2 L\sigma^2}{2} \leq \frac{(1 - \mu\beta)\beta^2 L\sigma^2}{2(\beta\mu - 1)} + \frac{\beta^2 L\sigma^2}{2} \leq 0,$$

we obtain

$$\begin{aligned}\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] &\leq \frac{\gamma + k - 1}{(\gamma + k)^2}\nu \\ &\leq \frac{\nu}{\gamma + k + 1},\end{aligned}$$

using $(\gamma + k)^2 \geq (\gamma + k + 1)(\gamma + k - 1) = (\gamma + k)^2 - 1$. Choosing $k = K - 1$ finally proves our result. □

From the result of Theorem 2.3.2, we see that choosing a decreasing stepsize results in a sublinear convergence rate, which is worse than the rate for stochastic gradient with constant stepsize. However, note that such a choice enables to reach any neighborhood of the optimal value.

Remark 2.3.2 *Choosing the stepsize (or, in machine learning language,* tuning the learning rate*) is one of the most critical issues in implementing stochastic gradient methods. As the rates above suggest, defining the stepsize according to the Lipschitz and strong convexity constants is critical to the performance of the method. In pratice, those constants are usual not known, and an estimation process must be performed: either a constant value is determined after a grid search, or the problem-dependent constants are estimated by a sampling procedure, and the step size is chosen according to these estimates.*

2.3.3 Analysis of stochastic gradient in the nonconvex case

Stochastic gradient (or some variant thereof) is the method of choice for training neural networks, which is usually a nonconvex problem, as illustrated in section 1.3.2. It is thus natural to ask whether global rates can be obtained for stochastic gradient in the nonconvex setting. For gradient descent, we know that one can guarantee (e.g. using a constant stepsize) that for any $K \geq 1$,

$$\min_{0 \leq k \leq K-1} \|\nabla f(\boldsymbol{x}_k)\| \leq \mathcal{O}\left(\frac{1}{\sqrt{K}}\right).$$

which is sometimes equivalently established as

$$\min_{0 \leq k \leq K-1} \|\nabla f(\boldsymbol{x}_k)\|^2 \leq \mathcal{O}\left(\frac{1}{K}\right).$$

As we will see, the guarantees that one can establish in the stochastic setting are affected by noise.

We begin by deriving a result in the context of constant stepsizes.

Theorem 2.3.3 *Let Assumptions 2.3.1 and 2.3.2 hold. Suppose that Algorithm 1 is run with a constant stepsize $\alpha_k = \alpha > 0$ where $\alpha \in \left(0, \frac{1}{L}\right]$. Then, for any $K \geq 1$,*

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=0}^{K-1} \|\nabla f(\boldsymbol{x}_k)\|^2\right] \leq \alpha L \sigma^2 + \frac{2(f(\boldsymbol{x}_0) - f^*)}{\alpha K}. \tag{2.3.8}$$

Proof. We again rely on the result of Proposition 2.3.1. For every index k, given that $\alpha_k = \alpha \leq \frac{1}{L}$, we have:

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] &\leq -\left(\alpha - \frac{\alpha^2 L}{2}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\alpha^2 L}{2}\sigma^2 \\ &\leq -\frac{\alpha}{2}\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\alpha^2 L}{2}\sigma^2.\end{aligned}$$

If we take the expectation over all indices, this relation becomes

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \quad \leq \quad -\frac{\alpha}{2}\,\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] + \frac{\alpha^2 L}{2}\sigma^2.$$

Using $f(\boldsymbol{x}_k) \geq f_{\text{low}}$ for all k and summing the above relation for $k = 0, \ldots, K-1$, we obtain

$$f_{\text{low}} - f(\boldsymbol{x}_0) \leq \mathbb{E}\left[f(\boldsymbol{x}_K) - f(\boldsymbol{x}_0)\right] \leq -\frac{\alpha}{2}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] + K\frac{\alpha^2 L}{2}\sigma^2.$$

The final result follows by re-arranging the terms. □

As in the strongly convex case, we see that the noise prevents from guaranteeing that the sum of squared gradients remains finite (which is the typical guarantee obtained for gradient descent). Note that the second term on the right-hand side of (2.3.8) corresponds to the usual (sublinear) convergence rate of gradient descent. The result of Theorem 2.3.3 thus guarantees that the average minimum gradient norm tends to concentrate in an interval defined by the noise level, as (2.3.8) implies

$$\lim_{K\to\infty}\mathbb{E}\left[\min_{0\leq k\leq K-1}\|\nabla f(\boldsymbol{x}_k)\|^2\right] \in \left[0, \alpha L\sigma^2\right].$$

In the case of decreasing stepsizes, we can provide the following guarantee.

Theorem 2.3.4 *Let Assumptions 2.3.1 and 2.3.2 hold. Suppose that Algorithm 1 is run with a decreasing stepsize sequence $\{\alpha_k\}$ such that $\alpha_k \in (0, \frac{1}{L}]$ for every k, and the sequence satisfies*

$$\sum_{k=0}^{\infty}\alpha_k = \infty \quad \textit{and} \quad \sum_{k=0}^{\infty}\alpha_k^2 < \infty.$$

Then, we have

$$\mathbb{E}\left[\frac{1}{\sum_{k=0}^{K-1}\alpha_k}\sum_{k=0}^{K-1}\alpha_k\|\nabla f(\boldsymbol{x}_k)\|^2\right] \to 0 \quad \textit{as} \quad K \to \infty. \tag{2.3.9}$$

Proof. The beginning of this proof is similar to that of Theorem 2.3.3. By Proposition 2.3.1, we have

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \quad &\leq \quad -\left(\alpha_k - \frac{\alpha_k^2 L}{2}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\alpha_k^2 L}{2}\sigma^2 \\ &\leq \quad -\frac{\alpha_k}{2}\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\alpha_k^2 L}{2}\sigma^2,\end{aligned}$$

and by taking the expectation over all indices $i_0, \ldots, i_k$, we get

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \quad \leq \quad -\frac{\alpha_k}{2}\,\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] + \frac{\alpha_k^2 L}{2}\sigma^2.$$

Summing this relation for every $k \in \{0, \ldots, K-1\}$ gives

$$\mathbb{E}\left[f(\boldsymbol{w}_K) - f(\boldsymbol{w}_0)\right] \quad \leq \quad -\frac{1}{2}\sum_{k=0}^{K-1}\alpha_k\,\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] + \frac{L\sigma^2}{2}\sum_{k=0}^{K-1}\alpha_k^2.$$

Using $f(\boldsymbol{x}_K) \geq f_{\text{low}}$, we obtain

$$\sum_{k=0}^{K-1} \alpha_k \, \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] \quad \leq \quad 2(f(\boldsymbol{x}_0) - f_{\text{low}}) + L\sigma^2 \sum_{k=0}^{K-1} \alpha_k^2.$$

By assumption, we know that $\sum_{k=0}^{K-1} \alpha_k^2 \leq \sum_{k=0}^{\infty} \alpha_k^2 < \infty$, thus the right-hand side is finite for every K. This implies that

$$\lim_{K\to\infty} \sum_{k=0}^{K-1} \alpha_k \, \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right] < \infty.$$

Thanks to our assumptions on $\{\alpha_k\}$, we also have $\sum_{k=0}^{K-1} \alpha_k \to \infty$ as $K \to \infty$. We can thus conclude that

$$\lim_{K\to\infty} \frac{1}{\sum_{k=0}^{K-1} \alpha_k} \sum_{k=0}^{K-1} \alpha_k \, \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|^2\right].$$

□

Theorem 2.3.4 shows that a weighted sum of squared gradients converges to zero regardless of the noise level (which could not be guaranteed with a constant stepsize). We can then derive the following corollaries.

Corollary 2.3.1 *Let the assumptions of Theorem 2.3.4 hold. For any $K \in \mathbb{N}$, let $k(K)$ be a random index chosen such that*

$$\mathbb{P}\left(k(K) = k\right) = \frac{\alpha_k}{\sum_{k=0}^{K-1} \alpha_k} \quad \forall k = 0, \ldots, K-1.$$

Then, $\|\nabla f(\boldsymbol{x}_{k(K)})\| \to 0$ in probability as $K \to \infty$, i.e.

$$\forall \epsilon > 0, \; \mathbb{P}\left(\|\nabla f(\boldsymbol{x}_{k(K)})\| \geq \epsilon\right) \to 0 \quad \text{as} \quad K \to \infty.$$

Proof. Let $\epsilon > 0$. Since $\mathbb{P}\left(\|\nabla f(\boldsymbol{x}_{k(K)})\| \geq \epsilon\right) = \mathbb{P}\left(\|\nabla f(\boldsymbol{x}_{k(K)})\|^2 \geq \epsilon^2\right)$, using Markov's inequality gives

$$\mathbb{P}\left(\|\nabla f(\boldsymbol{x}_{k(K)})\|^2 \geq \epsilon^2\right) \leq \frac{1}{\epsilon^2} \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_{k(K)})\|^2\right],$$

where the expectation is taken over both the randomness from the algorithm (selection of the stochastic gradient) and the randomness of the analysis (choice of $k(K)$). These two sources of randomness are independent, thus we write

$$\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_{k(K)})\|^2\right] = \mathbb{E}_{\{i_k\}}\left[\mathbb{E}_{k(K)}\left[\|\nabla f(\boldsymbol{x}_{k(K)})\|^2\right]\right],$$

where $\mathbb{E}_{\{i_k\}}[\cdot]$ is the randomness over the algorithm and $\mathbb{E}_{k(K)}[\cdot]$ is the randomness over $k(K)$.

$$\begin{aligned}
\mathbb{P}\left(\|\nabla f(\boldsymbol{x}_{k(K)})\|^2 \geq \epsilon^2\right) &\leq \frac{1}{\epsilon^2} \mathbb{E}_{\{i_k\}}\left[\mathbb{E}_{k(K)}\left[\|\nabla f(\boldsymbol{x}_{k(K)})\|^2\right]\right] \\
&= \frac{1}{\epsilon^2} \mathbb{E}_{\{i_k\}}\left[\sum_{k=0}^{K-1} \mathbb{P}\left(k(K) = k\right) \|\nabla f(\boldsymbol{x}_k)\|^2\right] \\
&= \frac{1}{\epsilon^2} \mathbb{E}_{\{i_k\}}\left[\sum_{k=0}^{K-1} \frac{\alpha_k}{\sum_{k=0}^{K-1} \alpha_k} \|\nabla f(\boldsymbol{x}_k)\|^2\right] \to 0 \quad \text{as } K \to \infty,
\end{aligned}$$

where the last observation comes from Theorem 2.3.4. □

Note that results similar to those above can be derived using a batch approach, with appropriate changes in the noise level. Those lead to the same observations as in the strongly convex case.

2.4 Concluding remarks

From a pure optimization perspective, stochastic gradient methods may not seem so attractive, as they only rely on partial information from the gradient and possess worse convergence guarantees than gradient descent. However, they have encountered tremendous success in data-related applications, where computing gradients involves looking at the entire data and is thus too prohibitive. On the contrary, using stochastic gradient estimates represents a significantly cheaper cost per iteration; in a data science setting, where there can be redundancies (or even underlying randomness) in the data, such updates do not necessarily hinder the progress of the algorithm, but rather lead to faster convergence in practice.

To end this section, we discuss several follow-ups on our analysis, both theoretical and practical.

Conditioning All our results heavily depend on the Lipschitz constant for the gradient (and, in the strongly convex case, on the strong convexity constant). Ill-conditioned problems, for which the ratio $\frac{L}{\mu}$ is much larger than 1, pose a significant challenge for stochastic gradient, as the speed of the method depends on this ratio. Similarly, a large Lipschitz constant enforces restrictions on the stepsize that may lead to small, and thus inefficient steps. The analysis can be refined by considering Lipschitz constants associated with every f_i, potentially defining larger values for the step size. Rescaling techniques, that transform the objective (either locally or globally), can also improve these constants and lead to dramatic speed-ups in practice.

Sharpness The rates achieved by stochastic gradient are sharp, in that under the same set of assumptions (and access to a stochastic gradient oracle), there does not exist a method that has a better rate than stochastic gradient (for instance, $\mathcal{O}(1/k)$ with a decreasing step size). Under additional assumptions on the problem, however, methods with improved complexity bounds can be developed.

Extension to the online setting We have presented the analysis in the case of the finite-sum problem (2.1.1), but it is also possible to generalize the analysis to stochastic optimization problem involving an expected value. Under relatively little assumptions about the problem and the method, one typically obtain convergence rates in expectation for convex and nonconvex problems.

Without gradients The above theory can be generalized to the nonsmooth setting, in which only subgradients of the component functions can be computed. In practice, subgradients can be obtained numerically, and in case of mild nonsmoothness (such as induced by the use of the ReLU activation function $t \mapsto \max\{0, t\}$), closed-form expressions for the subgradients can be available.

Chapter 3

Advanced concepts in stochastic gradient methods

In this chapter, we study more elaborate variants on the stochastic gradient paradigm. These methods have been studied extensively since they showed promise in deep learning. One of the main challenges in improving efficiency of these algorithms in practice lies in reducing the variance with respect to the stochastic gradients: we describe several techniques that provably reduce the variance in Section 3.1, while discussing their practical appeal. Other schemes aim at accelerating the performance of stochastic gradient through more sophisticated updates: we present examples of such techniques in Section 3.2.

3.1 Variance reduction techniques

As we saw in the previous section, the theory for stochastic gradient is based on Assumption 2.3.2, and in particular on the fact that the variance of stochastic gradient estimates is bounded (by σ^2). It can clearly be seen from bounds such as (2.3.5) that the bigger σ is, the looser the bound becomes. More practically, this means that gradient estimates with high variance are unlikely to yield fast convergence.

Variance reduction techniques have precisely been developed in the aim of diminishing the variance of traditional stochastic gradient estimates. They can be categorized in two families, that either exploit more sampled gradients at every iteration, or use past history of the method. In these notes, we will focus on the former category.

3.1.1 Using a batch size

Consider the finite-sum problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}). \tag{3.1.1}$$

The use of a **single** sample is partially responsible for the importance of the variance term σ^2 in Assumption 2.3.2. Recall for instance the result of Theorem 2.3.1:

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu} + (1-\alpha\mu)^k \left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu}\right]. \tag{3.1.2}$$

Suppose now that we use a (mini-)batch of such gradient estimates to construct our step. Because we are averaging more component gradients, we can expect the variance of such an estimate to be lower. This is formalized in the following proposition.

Proposition 3.1.1 *Under Assumptions 2.3.1 and 2.3.2, the variance of a mini-batch stochastic gradient estimate using n_b samples drawn with replacement satisfies*

$$\mathbb{E}_{S_k}\left[\left\| \tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right\|^2\right] \leq \frac{\sigma^2}{n_b} + \|\nabla f(\boldsymbol{x}_k)\|^2.$$

Proof. The last part of Assumption 2.3.2 can be rewritten as

$$\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{x}_k)\|^2\right] - \|\nabla f(\boldsymbol{x}_k)\|^2 \leq \sigma^2.$$

From the second part of Assumption 2.3.2, we can rewrite the left-hand side as

$$\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{x}_k)\|^2\right] - \|\,\mathbb{E}_{i_k}\left[\nabla f(\boldsymbol{x}_k)\right]\|^2,$$

that represents a variance term for the stochastic gradient[1]. Similarly, when considering a batch stochastic gradient estimate, we consider

$$\mathbb{E}_{S_k}\left[\left\| \tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right\|^2\right] - \left\|\mathbb{E}_{S_k}\left[\tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right]\right\|^2,$$

which again a variance term for a weighted sum of the random vectors $\{\nabla f_i(\boldsymbol{x}_k)\}_{i\in S_k}$. By assumption, those vectors are independent and identically distributed. Therefore, using the properties of the variance of a linear combination of i.i.d. variables, we obtain that:

$$\begin{aligned}\mathbb{E}_{S_k}\left[\left\| \tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right\|^2\right] - \left\|\mathbb{E}_{S_k}\left[\tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right]\right\|^2 &= \frac{1}{n_b^2} n_b \left(\mathbb{E}_i\left[\|\nabla f_i(\boldsymbol{x}_k)\|^2\right] - \|\,\mathbb{E}_i\left[\nabla f(\boldsymbol{x}_k)\right]\|^2\right) \\ &\leq \frac{1}{n_b}\sigma^2, \end{aligned} \tag{3.1.3}$$

where i is an arbitrary index following the distribution of the indices in S_k. Using again the second part of Assumption 2.3.2 gives

$$\mathbb{E}_{S_k}\left[\tfrac{1}{|S_k|}\sum_{i\in S_k} \nabla f_i(\boldsymbol{x}_k)\right] = \frac{1}{n_b} \times n_b\, \mathbb{E}_i\left[\nabla f_i(\boldsymbol{x}_k)\right] = \nabla f(\boldsymbol{x}_k).$$

Plugging this into (3.1.3) gives the desired result. □

As a result, if we use a mini-batch of size n_b instead of a single stochastic gradient, we can derive a result analogous to Theorem 2.3.1:

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu n_b} + (1-\alpha\mu)^k\left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L\sigma^2}{2\mu n_b}\right]. \tag{3.1.4}$$

[1] Usual practice in multidimensional statistics considers a covariance matrix that expresses the variance with respect to all pairs of coordinates, but the operator above can also be shown to act like a variance.

This result shows that for a given number of iterations and step size, a batch method will be able to reach a closer neighborhood of the optimal objective function than standard stochastic gradient. To achieve a similar result with Algorithm 1, one would need to choose $\frac{\alpha}{n_b}$ as a stepsize, so that (2.3.1) becomes

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu n_b} + \left(1 - \frac{\alpha\mu}{n_b}\right)^k \left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu n_b}\right],$$

This rate indicates that stochastic gradient would need n_b times more iterations than mini-batch SG to reach an equivalent function value, while employing a smaller stepsize. On the other hand, stochastic gradient iterations are n_b times cheaper than those of the mini-batch method. Besides, for any value α that satisfies the requirements of Theorem 2.3.1 for the batch method, the value $\frac{\alpha}{n_b}$ will satisfy the requirements of the theorem for the stochastic gradient method. The converse is not true, thus one cannot guarantee that a batch method can compensate its per-iteration cost by employing a bigger step than that of stochastic gradient.

Remark 3.1.1 *Note that the above result is valid for batch sizes with replacement. The results can be significantly different when the batches are drawn without replacement: for instance, if n batch components are drawn with replacement, one obtains a variance of at most $\frac{\sigma^2+\|\nabla f(\boldsymbol{x}_k)\|^2}{n}$, per Proposition 3.1.1, while if these n components are drawn without replacement, the resulting gradient estimate is the* exact *gradient, thus its variance is 0.*

Dynamic sampling Rather than using a constant batch size, one can consider a dynamical sample size that grows geometrically. This is possible for any finite dataset by allowing sampling with replacement, but this can be particularly useful on very large datasets or in an online setting where one can query as many examples as necessary at every iteration. Provided the resulting average is unbiased, and the number of examples grows at a geometric rate, it is possible to derive a sublinear rate of convergence for such a dynamic stochastic gradient approach. However, the use of such an approach remains elusive in pratice. More common techniques focus on choosing the batch size in an adaptive way, that can depend upon the behavior of the method (typically, increase the batch size if the algorithm appears to be stalling) or algorithmic quantities like second-order information (when available). These have proven useful in some learning applications, but remain out of computational reach for certain tasks like training very deep neural architectures.

3.1.2 Gradient aggregation methods

We now turn to **gradient aggregation** methods, that have attracted a lot of attention in the learning and optimization community because of the linear convergence rates that can be shown for such methods. Their main paradigm consists in computing a **full gradient step** at regular intervals, in order to correct high-variance components that could arise from the stochastic gradient update. Many variants on this idea have been proposed over the last decade; we review below the most significant ones, and provide an algorithmic sketch of these methods.

For the rest of this section, we assume that we are in the assumptions of Theorem 2.3.2 (in particular, the function f is $\mathcal{C}_L^{1,1}$ and μ-strongly convex). Recall that under these assumptions, we are able to show a sublinear rate of decrease for the function value, in $\mathcal{O}(\frac{1}{k})$.

SVRG The first gradient aggregation method we study is called SVRG, for *Stochastic Variance-Reduced Gradient*. It proceeds in cycles of m sub-iterations: at the beginning of every major iteration,

a full gradient $\nabla f(\boldsymbol{x}_k)$ is computed, then m iterations involving a single additional sampled gradient are performed. That is, we set $\tilde{\boldsymbol{x}}_0 = \boldsymbol{w}_k$ and $\tilde{\boldsymbol{x}}_{j+1} = \tilde{\boldsymbol{w}}_j - \alpha\tilde{\boldsymbol{g}}_j$, where

$$\tilde{\boldsymbol{g}}_k = \nabla f_{i_j}(\tilde{\boldsymbol{x}}_j) - \nabla f_{i_j}(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k). \tag{3.1.5}$$

The use of this estimate leads to a better bound for the variance of the stochastic gradient especially when $\tilde{\boldsymbol{x}}_j$ is close to $\boldsymbol{x}_k$. Moreover, one can show that it indeed achieves a linear rate in terms of iterations.

Theorem 3.1.1 *Under Assumption 2.3.1 and 2.3.3, suppose that the stepsize α and the length of the inner loop m used in Algorithm 2 satisfy*

$$\rho := \frac{1}{1-2\alpha L}\left(\frac{1}{m\mu\alpha} + 2L\alpha\right) \in (0,1).$$

Then,

$$\mathbb{E}\left[f(\boldsymbol{x}_k)\right] - f^* \leq \rho^k \left(f(\boldsymbol{x}_0) - f^*\right). \tag{3.1.6}$$

Algorithm 2: Basic SVRG method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$, $\alpha > 0$, $m \in \mathbb{N}$.
for $k = 0, 1 \ldots$ **do**
  Compute the full gradient $\nabla f(\boldsymbol{x}_k) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\boldsymbol{x}_k)$
  Set $\tilde{\boldsymbol{x}}_0 := \boldsymbol{x}_k$.
  for $j = 0, \ldots, m-1$ **do**
    Draw a random index i_j uniformly in $\{1, \ldots, n\}$.
    Set $\tilde{\boldsymbol{g}}_j := \nabla f_{i_j}(\tilde{\boldsymbol{x}}_j) - \nabla f_{i_j}(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)$.
    Set $\tilde{\boldsymbol{w}}_{j+1} := \tilde{\boldsymbol{x}}_j - \alpha\tilde{\boldsymbol{g}}_j$.
  end
  Draw j uniformly at random in $\{0, \ldots, m-1\}$ and set $\boldsymbol{x}_{k+1} = \tilde{\boldsymbol{x}}_{j+1}$.
end

One iteration of SVRG is comparable in cost to a full gradient iteration, because $2m + n$ gradients are required per iteration. However, it can still be faster than gradient descent, because of the intrinsic randomness.

Remark 3.1.2 *The SVRG method can be quite efficient in applications that require a high accuracy (i.e. $\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \epsilon$ with a small $\epsilon > 0$); however, for the first epochs, one generally notices that the stochastic gradient method is more efficient.*

SAGA Unlike SVRG, the SAGA method (derived from the Stochastic Average Gradient algorithm, or SAG) does not operate in cycles, and only requires one component gradient per iteration past the first one. It does, however, maintain a gradient estimate formed by n stochastic gradients *evaluated at different points throughout the optimization process*. Indeed, at every iteration, the

method has access to a value of every component gradient ∇f_i at some previous iterate $\boldsymbol{x}_{[i]}$. It then selects an index j at random and defines

$$\boldsymbol{g}_k := \nabla f_j(\boldsymbol{x}_k) - \nabla f_j(\boldsymbol{x}_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_{[i]}).$$

With standard techniques for choosing j, such as uniform sampling, one can show that $\mathbb{E}_j[\boldsymbol{g}_k] = \nabla f(\boldsymbol{x}_k)$, and the variance of this estimator is lower than that of a classical stochastic gradient. This is the key idea behind showing that this method also converges linearly.

Algorithm 3: SAGA method.

Initialization: $\boldsymbol{x}_0 \in \mathbb{R}^d$, $\alpha > 0$.
for $i = 1, \dots, n$ **do**
  Compute $\nabla f_i(\boldsymbol{x}_0)$.
  Set $\nabla f_i(\boldsymbol{x}_{[i]}) := \nabla f_i(\boldsymbol{x}_0)$.
end
for $k = 0, 1 \dots$ **do**
  Draw a random index $j \in \{1, \dots, n\}$.
  Compute $\nabla f_j(\boldsymbol{x}_k)$.
  Set $\boldsymbol{g}_k := \nabla f_j(\boldsymbol{x}_k) - \nabla f_j(\boldsymbol{x}_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_{[i]})$.
  Update $\nabla f_j(\boldsymbol{x}_{[j]}) = \nabla f_j(\boldsymbol{x}_k)$.
  Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k$.
end

The typical rate of SAGA is given below, in terms of convergence to the iterates.

Theorem 3.1.2 *Under Assumptions 2.3.1 and 2.3.3, suppose that the stepsize of Algorithm 3 is chosen as* $\alpha = \frac{1}{2(\mu n + L)}$. *Then,*

$$\mathbb{E}\left[\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2\right] \leq \left(1 - \frac{\mu}{2(\mu n + L)}\right)^k \left(\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \frac{n(f(\boldsymbol{x}_0) - f^*)}{\mu n + L}\right). \tag{3.1.7}$$

Note that the strong convexity constant is not needed here, as a step of $\alpha = \frac{1}{3L}$ would also yield linear convergence.

The rate of Theorem 3.1.2 can be shown to be of the same order than that of SVRG, but the per-iteration cost of SAGA is comparable to that of the classical stochastic gradient method. The main drawback of SAGA is that it requires to store n gradient vectors to be able to perform its update, which can be prohibitive in many settings (this is however affordable in certain problems such as logistic and least-squares regression problems).

Remark 3.1.3 *Despite their strong guarantees, gradient aggregation methods have not been widely exploited in practice, due to the cost of full gradient evaluations, that can remain too prohibitive for certain applications. In particular, variance reduction methods can be inefficient for training of neural network architectures; on the other hand, promising results have been obtained in other settings such as reinforcement learning.*

3.1.3 Iterate averaging

The iterates of a stochastic gradient sequence can be observed to oscillate around minimizers: this motivated the use of averaging techniques to limit this oscillating behavior. The basic idea consists in maintaining a sequence of average iterates during a run of stochastic gradient. Considering our basic stochastic gradient setup with uniformly sampled indexes, this would result in the following iteration

$$\left\{ \begin{array}{rcl} \boldsymbol{x}_{k+1} & = & \boldsymbol{x}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{x}_k) \\ \hat{\boldsymbol{x}}_{k+1} & = & \frac{1}{k+1} \sum_{j=0}^{k} \boldsymbol{x}_j. \end{array} \right. \tag{3.1.8}$$

This averaging process has been widely used in stochastic approximation methods and in stochastic programming, leading to so-called *ergodic convergence rates*. Under appropriate assumptions on the stepsize and the objective function, similar rates can be established for the recursion (3.1.8). Moreover, the average is provably a more robust solution than the last iterate returned by the method. Overall, with careful parameter selection, averaging can prove to be a powerful paradigm; note that, in practice, maintaining such an average iterate can be prone to cancellation or numerical errors.

3.2 Stochastic gradient methods for deep learning

Although it is common to observe good performance of stochastic gradient compared to gradient descent, deploying a stochastic gradient method so as to be efficient on very large-scale applications poses a number of challenges. In this section, we review the main stochastic gradient techniques that are used to train deep learning models. Our focus remains on finite-sum problems of the form (2.1.1) under Assumption 2.3.1. Our goal is to study several variants on the iterative scheme

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k, \tag{3.2.1}$$

where $\alpha > 0$ is a fixed stepsize (or learning rate), and $\boldsymbol{g}_k$ is a stochastic gradient estimator, that may correspond to taking a single term from the finite sum (as in stochastic gradient) or to a batch of indices.

To encompass all the variants of interest, we consider a more general iteration of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{m}_k \oslash \boldsymbol{v}_k, \tag{3.2.2}$$

where $\alpha > 0$, $\boldsymbol{m}_k, \boldsymbol{v}_k \in \mathbb{R}^d$ and $\oslash$ denotes the componentwise division operator:

$$\boldsymbol{m}_k \oslash \boldsymbol{v}_k := \left[\frac{[\boldsymbol{m}_k]_i}{[\boldsymbol{v}_k]_i} \right]_{i=1,\dots,d}.$$

To see that (3.2.2) generalizes (3.2.1), set $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$. The iteration (3.2.2) is particularly convenient to express the popular methods in deep learning using a single format.

3.2.1 Stochastic gradient with momentum

Most practical implementations of stochastic gradient combine the basic step (3.2.1) together with a momentum term, similarly to accelerate methods in deterministic optimization[2]. An iteration of

[2] See Irène Waldspurger's lectures for more details.

gradient descent with momentum reads

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha(1-\beta)\boldsymbol{g}_k + \alpha\beta\left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right), \tag{3.2.3}$$

where $\beta \in (0,1)$ is a momentum parameter (when $\beta = 0$ we again obtain the standard stochastic gradient iteration). Iteration (3.2.3) can be seen as a version of Polyak's method where the gradient-type step is combined with the previous direction. As for the heavy-ball method, the idea is to incorporate information from the previous step through momentum. In practice, the iteration (3.2.3) often leads to accumulation of good steps (in terms of optimization), whereas bad directions and bad steps tend to cancel out.

The basic method (3.2.3) can be recovered from (3.2.2), by setting $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$ and defining $\boldsymbol{m}_k$ in a recursive manner through $\boldsymbol{m}_{-1} = \mathbf{0}_{\mathbb{R}^d}$ and

$$\boldsymbol{m}_k = (1-\beta)\boldsymbol{g}_k - \beta\boldsymbol{m}_{k-1} \quad \forall k \in \mathbb{N}.$$

where $\beta \in (0,1)$.

Stochastic gradient with momentum is implemented in most deep learning librairies such as PyTorch. It has shown great success in training deep neural networks on computer vision problems, and is partly responsible for the rise of deep learning in the early 2010s.

Remark 3.2.1 *Theoretical guarantees for the iteration* (3.2.3) *are much more difficult to obtain than for accelerated gradient (in particular, it is not well understood whether such a method can be provably faster than SGD). Nevertheless, stochastic gradient techniques with momentum are widely used in practice, even on nonconvex problems such as neural network training.*

3.2.2 AdaGrad

The adaptive gradient method, or AdaGrad, was proposed in 2011 to address the issue of setting the learning rate α in stochastic gradient. Rather than using costly procedures such as line search, AdaGrad scales every coordinate of the stochastic gradient using information from the values of that coordinate in the previous iterations. Mathematically, the method maintains a sequence $\{\boldsymbol{r}_k\}_k$ defined by

$$\forall i = 1,\dots,d, \quad \begin{cases} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = [\boldsymbol{r}_{k-1}]_i + [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{cases} \tag{3.2.4}$$

The AdaGrad iteration can then be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k \oslash \sqrt{\boldsymbol{r}_k}, \tag{3.2.5}$$

where the square root is applied componentwise to $\boldsymbol{r}_k$. This iteration is a special cas of (3.2.2), where $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$. The novelty in AdaGrad does not lie in the use of momentum, but in the use of one stepsize per coordinate. The stepsize sequence thus has the form

$$\left\{ \left[\frac{\alpha}{\sqrt{[\boldsymbol{r}_k]_i}} \right]_{i=1}^{d} \right\}_k .$$

The resulting *diagonal scaling* on the coordinates of $\boldsymbol{g}_k$ leads to stepsizes that adapt to coordinates that can vary by orders of magnitude (which would require a careful choice of α in basic stochastic gradient). On the other hand, the accumulation process at work in the definition of $\boldsymbol{r}_k$ results in stepsizes that are monotonically decreasing, and that often converge quickly towards 0.

Remark 3.2.2 *In practice, $\boldsymbol{r}_k$ is replaced by $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ where $\eta > 0$ is a small value that helps with numerical stability.*

The AdaGrad method is particularly interesting for problems with sparse gradients, in which stochastic gradients tend to have many zero coordinates. In this situation, using $\boldsymbol{r}_k$ will only modify the stepsizes corresponding to nonzero gradient coordinates. Many problems in recommendation systems have a sparse structure, and AdaGrad is considered to be an efficient method for this class of problems.

3.2.3 RMSProp

The RMSProp (*Root Mean Square Propagation*) algorithm is similar in spirit to AdaGrad, in that it scales the gradient components at every step. This method relies on a sequence $\{\boldsymbol{r}_k\}_k$ defined by

$$\forall i = 1, \ldots, d, \quad \begin{cases} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = (1-\lambda)[\boldsymbol{r}_{k-1}]_i + \lambda [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{cases} \tag{3.2.6}$$

where $\lambda \in (0,1)$. The value of λ is used to put more weight on the current gradient coordinates than on the coordinates from past iterations (this information being contained in $\boldsymbol{r}_{k-1}$). This simple idea slows down the decrease of the stepsizes to 0, compared to the stepsizes of AdaGrad.

With the definition (3.2.6), the RMSProp iteration corresponds has the same form as that of AdaGrad, that is, a special case of (3.2.2) with $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$.

Remark 3.2.3 *As for AdaGrad, standard practice replaces $\boldsymbol{r}_k$ by $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$, where $\eta > 0$ is a small quantity.*

The RMSProp method has been found quite successful for training very deep neural networks.

3.2.4 Adam

The Adam optimization method was proposed in 2013. This method can be thought as combining the idea of momentum (used in the stochastic gradient method of Section 3.2.1) with the diagonal scaling procedure on which both AdaGrad and RMSProp are based. An iteration of Adam falls into the generic scheme (3.2.2) by setting

$$\boldsymbol{m}_k = \frac{(1-\beta_1)\sum_{j=0}^k \beta_1^{k-j} \boldsymbol{g}_j}{1-\beta_1^{k+1}} \tag{3.2.7}$$

and

$$\boldsymbol{v}_k = \sqrt{\frac{(1-\beta_2)\sum_{j=0}^k \beta_2^{k-j} \boldsymbol{g}_j \odot \boldsymbol{g}_j}{1-\beta_2^{k+1}}}. \tag{3.2.8}$$

Here $\beta_1, \beta_2 \in (0,1)$, and $\odot$ denotes the Hadamard or componentwise product

$$\boldsymbol{g}_k \odot \boldsymbol{g}_k = \left[[\boldsymbol{g}_k]_i^2 \right]_{i=1}^d .$$

Remark 3.2.4 *In practice, $\boldsymbol{v}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ (with small $\eta > 0$) is used in lieu of $\boldsymbol{v}_k$.*

The above formulas describe the two components of ADAM. On one hand, a weighted combination of the previous steps that puts the emphasis on the oldest steps (that have been least affected by momentum) defines the direction of the next step. On the other hand, a diagonal scaling is applied to the coordinates of this direction, again according to a weighted average of the coordinates from the previous iterations. This important feature, that has a statistical motivation, appears to be responsible for the success of ADAM in practice. The impressive performance of ADAM on training neural networks has contributed to its popularity, and it remains the preferred method today in numerous applications. In particular, ADAM and its variant ADAMW (based on regularization principle) are quite efficient on natural language processing models.

3.3 Using stochastic gradient in practice

The practical implementation of stochastic gradient methods varies significantly depending on the application/the problem at hand. As we mention several times in these notes, each machine learning task has its own characteristics, and a given variant of stochastic gradient may work well on one problem and stall on another. Below are a few (non-exhaustive) pieces of advice that one could try to guide their testing.

- A basic stochastic gradient method with carefully tuned constant step size can often provide very satisfactory results; ideally one chooses the largest stepsize that does not lead to divergence, through a grid search, or a more sophisticated procedure that involves information about the problem (e.g. Lipschitz constant).
- In a large-scale learning context (i.e. with a lot of training examples, but not necessarily a model as complex as a neural network), it is generally beneficial to consider a mini-batch version of stochastic gradient. A trade-off must be found between the cost of computing a batch gradient estimate and the possible gain in convergence speed.
- ADAM is one the preferred variants of stochastic gradient used in deep learning, which represents the state of the art: it is likely the method one would try first on a deep learning problem, as it has been efficiently implemented in popular packages like PYTORCH.
- Diagonal scaling helps with ill-conditioned problems, and is a powerful paradigm in deep neural architectures (RMSPROP is generally efficient in this setting).
- Any sparsity pattern (in the gradients/in the data) can help perform more efficient calculations. In terms of algorithms, this allows variants like ADAGRAD to perform well.
- Momentum can be quite useful in practice, but it requires to tune additional parameters, which can be cumbersome: similar observations can be made for accelerated techniques.
- Reducing the intrinsic noise of stochastic gradient is an important concern, that can be efficiently addressed in practice via iterate averaging.
- Variance reduction techniques can lead to very efficient methods, but their implementation cost makes them less suitable for very large problems and/or problems that require only low accuracy estimates for the solution.

Chapter 4

Exercises

Exercise 1: Recap on stochastic gradient

We consider a finite-sum optimization problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}), \tag{4.0.1}$$

where each function f_i depends on a single item of a dataset consisting in n elements. We suppose that every function f_i is continuously differentiable, and we define a family of algorithms methods based on a starting point $\boldsymbol{x}_0 \in \mathbb{R}^d$ as well as the recursion

$$\forall k \geq 0, \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k, \tag{4.0.2}$$

where $\alpha > 0$ is a given stepsize and $\boldsymbol{g}_k$ is an estimate of the gradient $\nabla f(x_k)$.

a) How should $\boldsymbol{g}_k$ be chosen in order for the recursion (4.0.2) to correspond to an instance of:
   i) gradient descent?
   ii) stochastic gradient?

b) Recall the definition of an epoch: what is the equivalent of this unit in terms of:
   i) iterations of gradient descent?
   ii) iterations of stochastic gradient?

c) We now focus on using of batch variants of stochastic gradient. Given a batch size $n_b \in \{1, \ldots, n\}$, we draw a batch index set $\mathcal{S}_k \subseteq \{1, \ldots, n\}$ based on the following distribution :

$$\forall \mathcal{S} \subseteq \{1, \ldots, n\}, \quad \mathbb{P}\left(\mathcal{S}_k = \mathcal{S}\right) := \begin{cases} \frac{1}{\binom{n}{n_b}} = \frac{n_b!(n-n_b)!}{n!} & \text{if } |\mathcal{S}| = n_b \\ 0 & \text{otherwise.} \end{cases} \tag{4.0.3}$$

We then set $\boldsymbol{g}_k = \frac{1}{|\mathcal{S}_k|}\sum_{i\in\mathcal{S}_k} \nabla f_i(\boldsymbol{x}_k)$ in the recursion (4.0.2).

   i) Show that $\mathbb{E}_{\mathcal{S}_k}\left[g_k\right] = \nabla f(x_k)$.

ii) Suppose that the function f is strongly convex; in that case, and under the appropriate assumptions on the problem, one can show that

$$\lim_{k\to\infty} \mathbb{E}\left[f(x_k) - f^*\right] \in \left[0, c\alpha\frac{M^2}{n_b}\right], \tag{4.0.4}$$

where $c > 0$ is a problem-dependent constant and $M^2 > 0$ is a bound on $\|\nabla f_i(\boldsymbol{x})\|$ for any $i \in \{1, \dots, n\}$ and $\boldsymbol{x} \in \mathbb{R}^d$. What property of (batch) stochastic gradient methods does this result illustrate?

d) Describe two modifications of the algorithm (among those covered in the lectures) that can lead to a guarantee of the form $\lim_{k\to\infty} \mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] = 0$.

e) *Practical situation :* Suppose that we want to compare stochastic gradient for several values of the batch size. While running on a given problem, we observe that $n_b = 1$ gives better convergence than $n_b = n$, while increasing the batch size from $n_b = 1$ to $n_b = n/10$ consistently improves the results, in that the method converges faster and to a smaller value of f. We then observe that the convergence slows down as we increase the batch size from $n/10$ to n. How can you explain these observations?

Exercise 2 : Importance sampling

We consider the finite-sum problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}), \tag{4.0.5}$$

where for every $i = 1, \dots, n$, the function f_i is $\mathcal{C}^1$ and its gradient is L_i-Lipschitz continuous. We also assume that the function f is μ-strongly convex. In what follows, let $c_i = \frac{nL_i}{\sum_{j=1}^n L_j}$.

We consider a variant on the framework of Algorithm 1, where we suppose that the random indices i_k are drawn according to an *importance sampling distribution* defined by

$$\forall i \in \{1, \dots, n\}, \qquad \mathbb{P}\left(i_k = i\right) \;=\; \frac{c_i}{\sum_{j=1}^n c_j}. \tag{4.0.6}$$

In addition, we suppose that the update 2.2.1 is replaced by

$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \frac{\alpha_k}{c_{i_k}}\nabla f_{i_k}(\boldsymbol{x}_k).$$

a) What can be the interest of such a strategy?

b) Show that $\mathbb{E}_{i_k}\left[\frac{1}{c_{i_k}}\nabla f_{i_k}(\boldsymbol{x}_k)\right] = \nabla f(\boldsymbol{x}_k)$.

c) It can be shown that ∇f is L-Lipschitz continuous with $L = \frac{1}{n}\sum_{i=1}^n L_i$. Suppose that we select a constant stepsize $\alpha_k = \frac{1}{L}\forall k$. Given a sampled index i_k, we wish to compare the classical stochastic gradient iteration to iteration (4.0.6).

i) Show that $\frac{\alpha_k}{c_{i_k}} = \frac{1}{L_{i_k}}$.

ii) When can we get $\frac{\alpha_k}{c_{i_k}} \geq \alpha_k$? What does this imply on the iteration (4.0.6) ?

d) Suppose that we know a Lipschitz constant L on ∇f with $L \geq \max_{1 \leq i \leq n} L_i$: with constant stepsize $\alpha \leq \frac{1}{L}$, we know that convergence guarantees can be obtained. How can the knowledge of the $\{L_i\}$s allow for a larger stepsize?

Exercise 3 : Batch methods

Under the assumptions used to obtain (3.1.4), show that one can actually derive the more precise result:

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu(2n_b - 1)} + \left(1 - \alpha\mu(2 - \tfrac{1}{n_b})\right)^k \left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu(2n_b - 1)}\right]. \tag{4.0.7}$$

How does this illustrate the strongest requirements on α that can be necessary for the batch methods?

Bibliography

[1] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Rev.*, 60:223–311, 2018.

[2] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, United Kingdom, 2019.

[3] J. C. Duchi. Introductory lectures on stochastic optimization. In A. C. Gilbert M. W. Mahoney, J. C. Duchi, editor, *The mathematics of data*, number 25 in IAS/Park City Mathematics Series. AMS, IAS/Park City Mathematics Institute, and Society for Industrial and Applied Mathematics, Princeton, 2018.

[4] R. M. Gower, M. Schmidt, F. Bach, and P. Richtárik. Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108:1968–1983, 2020.

[5] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning*. Cambridge University Press, Cambridge, United Kingdom, 2014.

[6] S. J. Wright. Optimization algorithms for data analysis. In A. C. Gilbert M. W. Mahoney, J. C. Duchi, editor, *The mathematics of data*, number 25 in IAS/Park City Mathematics Series. AMS, IAS/Park City Mathematics Institute, and Society for Industrial and Applied Mathematics, Princeton, 2018.

Appendix A

Basics in probability and statistics

A.1 Probability theory

The concept of probability originates from measure theory. All results in probability and statistics implicitly rely on probability spaces, i.e. triplets $(\Omega, \mathcal{A}, \mathbb{P})$, where

- Ω is a set of possible values, or outcomes;
- $\mathcal{A}$ is a family of subsets of Ω called set of events, that satisfy certain properties that make it a σ-algebra;
- $\mathbb{P} : \mathcal{A} \to [0, 1]$ is a probability measure, that satisfies in particular $\mathbb{P}(\emptyset) = 0$ and $\mathbb{P}(\Omega) = 1$.

Given this definition, a random variable is a mapping from a probability space to another space that induces a new probability measure on the latter. The term *random variable* is often used for scalar quantities, thus we will make a distinction between random variables and random vectors defined as follows:

- random variables z defined on a probability space $(\mathbb{R}, \mathcal{B}(\mathbb{R}), \mathbb{P})$ by

$$\forall B \in \mathcal{B}(\mathbb{R}), \quad \mathbb{P}(z \in B) = \mathbb{P}(B);$$

- random vectors $\boldsymbol{z} = \begin{bmatrix} z_1 \\ \cdots \\ z_d \end{bmatrix}$ of size d, defined on the probability space $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d), \mathbb{P})$.

In both case, the set of events will be the Borel σ-algebra $\mathcal{B}(\mathbb{R}^d)$.

A.2 Random variables

Although a generic study of random variables can be performed by considering them as taking a continuum of values, we begin by providing the more elementary definition of discrete random variables.

Definition A.2.1 (Discrete random variable) *A discrete random variable z is defined by*

- *A discrete set of possible values* $\mathcal{Z} = \{z_i\} \subset \mathbb{R}$*;*
- *An associated set of probabilities* $p = \{p_i\}$ *such that* $p_i \geq 0$, $\sum_i p_i = 1$ *and*

$$\forall \mathcal{S} \subset \mathcal{Z}, \quad \mathbb{P}(z \in \mathcal{S}) = \sum_{z_i \in \mathcal{S}} p_i.$$

Definition A.2.2 (Continuous random variable) *A continuous random variable* z *is defined by*

- *A continuous set of possible values* $\mathcal{Z} \subset \mathbb{R}$*;*
- *An associated probability density* $p : \mathcal{Z} \to \mathbb{R}^+$ *such that* $\int_{\mathbb{R}} p(z)\, dz = 1$ *and*

$$\forall \mathcal{S} \subset \mathcal{Z}, \quad \mathbb{P}(z \in \mathcal{S}) \; = \; \int_{z \in \mathcal{S}} p(z)\, dz.$$

For both continuous and discrete random variables, we will say that z follows a distribution characterized by $(p, \mathcal{Z})$, or simply p when the set of possible values is implicit from the definition of p.

To understand the behavior of random variables, one can look at the moments of their distribution (provided they are well defined). The canonical example of such a quantity is the mean (also called the expected value) of a random variable.

Definition A.2.3 (Expected value/Mean) *Let* z *be a random variable with a distribution* $(p, \mathcal{Z})$*, which we indicate as* $z \sim p$*. The* **expected value of** z *is defined by*

$$\mathbb{E}[z] = \mathbb{E}_z[z] = \begin{cases} \sum_{z_i \in \mathcal{Z}} z_i\, p(z = z_i) & \text{(discrete case)} \\ \\ \int_{\mathcal{Z}} z\, p(z)\, dz & \text{(continuous case).} \end{cases}$$

The expected value has several desirable properties that facilitate its use, especially the following.

Proposition A.2.1 *The expected value is a linear operator: that is, for every random variable* z *and every* $\alpha, \beta \in \mathbb{R}$*, one has:*

$$\mathbb{E}[\alpha\, z + \beta] = \alpha\, \mathbb{E}[z] + \beta;$$

The expected

Definition A.2.4 (Variance and standard deviation) *Let* z *be a random variable.*

- *The variance of* z *is defined by*

$$\mathrm{Var}[z] = \mathbb{E}\left[z^2\right] - \mathbb{E}[z]^2.$$

- *The standard deviation of* z *is the square root of the variance.*

Lemma A.2.1

- *If* z *is a discrete random variable, then* $\mathrm{Var}[z] = \sum_i p_i z_i^2 - \left[\sum_i p_i z_i\right]^2$*;*
- *If* z *has zero mean, i.e.* $\mathbb{E}[z] = 0$*, then* $\mathrm{Var}[z] = \mathbb{E}\left[z^2\right]$*.*

A.3 Pair of random variables

When two random variables possess the same distribution on the same probability space, we say that those variables are **identically distributed**. In a general setting, one can study the distribution of the pair formed by two random variables.

Definition A.3.1 (Joint distribution (discrete case)) *Let z and w be two discrete random variables taking values in $\mathcal{Z} = \{z_i\}$ and $\mathcal{W} = \{w_j\}$, respectively. The distribution of the pair of random variables (z, w) is defined by*

- *The set of possible values $\mathcal{Z} \times \mathcal{W} = \{(z_i, w_j)\}$;*
- *The discrete probability density $p = \{p_{i,j}\}$, where*

$$p_{i,j} = \mathbb{P}(z = z_i, w = w_j).$$

Definition A.3.2 (Joint distribution (continuous case)) *Let z and w be two continuous random variables taking values in $\mathcal{Z}$ and $\mathcal{W}$. The distribution of the pair of random variables (z, w) is defined by*

- *The set of possible values $\mathcal{Z} \times \mathcal{W}$;*
- *The continuous probability density $p : \mathcal{Z} \times \mathcal{W} \to \mathbb{R}^+$ such that*

$$\int_z \int_w p(z, w)\, dz\, dw \;=\; 1.$$

In the above definitions, we started from two random variables to obtain the joint distribution of the pair formed by these variables. It is also possible to go the other way around, by defining marginal laws.

Definition A.3.3 (Marginal laws (discrete case)) *Let z and w be two discrete random variables taking values in $\mathcal{Z} = \{z_i\}$ and $\mathcal{W} = \{w_j\}$, respectively. Let $\{p_{i,j}\}$ be the joint distribution of (z, w).*

- *The marginal law of z is given by $\{p_{i\bullet}\}_i$, where*

$$p_{i\bullet} := \mathbb{P}(z = z_i) = \sum_{j | w_j \in \mathcal{W}} \mathbb{P}(z = z_i,\ w = w_j) = \sum_j p_{i,j}.$$

- *Similarly, the marginal law of w is given by $\{p_{\bullet j}\}_j$, where*

$$p_{\bullet j} := \mathbb{P}(w = w_j) = \sum_{i | z_i \in \mathcal{Z}} \mathbb{P}(z = z_i,\ w = w_j) = \sum_i p_{i,j}.$$

Definition A.3.4 (Marginal laws (continuous case)) *Let z and w be two continuous random variables taking values in $\mathcal{Z}$ and $\mathcal{W}$, respectively. Let $p : (z, w) \mapsto p(z, w)$ be the joint density of (z, w).*

- *The marginal law of z, denoted by p_z or $p(z, \bullet)$, is the function $p_z : \mathcal{Z} \to \mathbb{R}^+$ given by*

$$\forall z \in \mathcal{Z}, \quad p_z(z) = \int_{\mathcal{W}} p(z, w)\, dw.$$

- *The marginal law of w, denoted by p_w or $p(\bullet, w)$, is the function $p_w : \mathcal{W} \to \mathbb{R}^+$ given by*

$$\forall w \in \mathcal{W}, \quad p_w(w) = \int_{\mathcal{Z}} p(z, w)\, dz.$$

Definition A.3.5 (Covariance and correlation) *Let z and w be two random variables. The covariance of z and w is defined by*

$$\mathrm{Cov}\,[z, w] = \mathbb{E}_{z,w}\left[(z - \mathbb{E}\,[z])\,(w - \mathbb{E}\,[w])\right].$$

The correlation of z and w is

$$\mathrm{Corr}\,[z, w] = \frac{\mathrm{Cov}\,[z, w]}{\sqrt{\mathrm{Var}_z\,[z]}\sqrt{\mathrm{Var}_w\,[w]}}.$$

Independent random variables Independence is widely used in statistics, where it is often combined with the notion of identically distributed variables: we then say that the random variables are **i.i.d.**, which stands for "independent, identically distributed".

Definition A.3.6 (Independent variables) *Let z and w be two random variables with distributions $(p_z, \mathcal{Z})$ and $(p_w, \mathcal{W})$, respectively. The variables z and w are called* **independent** *if the pair (z, w) satisfies*

$$\forall \mathcal{S} \times \mathcal{T} \subset \mathcal{Z} \times \mathcal{W}, \quad \mathbb{P}\left(z \in \mathcal{S}, w \in \mathcal{T}\right) \;=\; \mathbb{P}\left(z \in \mathcal{S}\right)\mathbb{P}\left(w \in \mathcal{T}\right).$$

Independence allows for an easy characterization of the joint distribution, as illustrated by the following result.

Proposition A.3.1 *Let z and w be two independent random variables. Then, their joint distribution is obtained as the product of the marginal distributions. We thus have*

$$\begin{cases} p_{ij} = p_{i\bullet} \times p_{\bullet j} & \textit{(discrete case)} \\ p(z, w) = p_z(z) \times p_w(w) & \textit{(continuous case).} \end{cases}$$

Proposition A.3.2 *Let z and w be two independent random variables. Then, these values are decorrelated, i. e. $\mathrm{Cov}\,[z, w] = \mathrm{Corr}\,[z, w] = 0$.*

A.4 Multidimensional statistics

Most of the previous results on random variables can be extended to the case of **random vectors**, i.e. multidimensional random quantities. We provide below the basic concepts.

Definition A.4.1 (Law of a random vector) *Let $\boldsymbol{z} = [z_i]_i$ be a random vector in $\mathbb{R}^n$: the law (or the distribution) of $\boldsymbol{z}$ is given by the joint distribution of its components. In particular, we define the following moments of this distribution:*

- *the expected value of $\boldsymbol{z}$ is the vector of the expected values of each component:*

$$\mathbb{E}[\boldsymbol{z}] = \{\mathbb{E}[z_i]\}_i \in \mathbb{R}^n;$$

  where the expected value is taken with respect to $\boldsymbol{z}$;

- *the* **covariance** *matrix of $\boldsymbol{z}$, denoted by $\mathrm{Var}[\boldsymbol{z}]$ or $\boldsymbol{\Sigma}_{\boldsymbol{z}}$ is the matrix of the covariances between each component*

$$\forall 1 \leq i, j \leq n, \quad [\boldsymbol{\Sigma}_{\boldsymbol{z}}]_{i,j} := \mathbb{E}[(z_i - \mathbb{E}[z_i])(z_j - \mathbb{E}[z_j])].$$

Note that the covariance matrix can be written as

$$\boldsymbol{\Sigma}_{\boldsymbol{z}} = \mathbb{E}\left[(\boldsymbol{z} - \mathbb{E}[\boldsymbol{z}])(\boldsymbol{z} - \mathbb{E}[\boldsymbol{z}])^{\mathrm{T}}\right] \in \mathbb{R}^{n \times n}.$$

Lemma A.4.1 *If the components of a random vector are independent, then its covariance matrix is diagonal.*

A.5 Markov's inequality

Many statistical results rely on providing bounds on probability levels, or moments of a given random variable. One of the most prominent results in this respect, due to Markov, is given below.

Theorem A.5.1 (Markov's inequality) *Let x be a nonnegative random variable and $\epsilon > 0$. Then,*

$$\mathbb{P}(x \geq a) \leq \frac{\mathbb{E}[x]}{a}.$$

Appendix B

Solutions of the exercises

Solutions of Exercise 1: Recap on stochastic gradient

a) In $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k$, the vector $\boldsymbol{g}_k$ should be set as

   i) $\nabla f(\boldsymbol{x}_k)$ for the method to be an instance of gradient descent;

   ii) $\nabla f_{i_k}(\boldsymbol{x}_k)$, with i_k drawn at random in $\{1, \dots, n\}$, to be an instance of stochastic gradient.

b) An epoch is a unit of cost equivalent to n accesses to data points. As a result, this unit corresponds to

   i) 1 iteration of gradient descent, in which all n data points need to be accessed in order to compute the gradient;

   ii) n iterations of stochastic gradient, since every iteration of that form only requires access to 1 data point. Equivalently, an iteration of stochastic gradient corresponds to $\frac{1}{n}$ epoch.

c) (Batch variant.)

   i) By definition of the expected value, one has

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}_k}[\boldsymbol{g}_k] &= \sum_{\mathcal{S} \subseteq \{1,\dots,n\}} \mathbb{P}(\mathcal{S}_k = \mathcal{S}) \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla_i f(\boldsymbol{x}_k) \\
&= \sum_{\substack{\mathcal{S} \subseteq \{1,\dots,n\} \\ |\mathcal{S}| = n_b}} \frac{1}{\binom{n}{n_b}} \frac{1}{n_b} \sum_{i \in \mathcal{S}} \nabla_i f(\boldsymbol{w}_k) \\
&= \frac{1}{\binom{n}{n_b}} \frac{1}{n_b} \sum_{\substack{\mathcal{S} \subseteq \{1,\dots,n\} \\ |\mathcal{S}| = n_b}} \sum_{i \in \mathcal{S}} \nabla_i f(\boldsymbol{x}_k)
\end{aligned}$$

   The random set $\mathcal{S}_k$ takes $\binom{n}{n_b}$ possible values. If we consider any index $i \in \{1, \dots, n\}$, this index appears in exactly $\binom{n-1}{n_b-1}$ index sets of cardinality n_b out of the possible $\binom{n}{n_b}$. Therefore,

$$\sum_{\substack{\mathcal{S} \subseteq \{1,\dots,n\} \\ |\mathcal{S}| = n_b}} \sum_{i \in \mathcal{S}} \nabla_i f(\boldsymbol{x}_k) = \binom{n-1}{n_b - 1} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_k).$$

Using $\binom{n}{n_b} = \frac{n}{n_b}\binom{n-1}{n_b-1}$, we thus obtain

$$\begin{aligned}\mathbb{E}_{\mathcal{S}_k}[\boldsymbol{g}_k] &= \frac{1}{\binom{n}{n_b}} \frac{1}{n_b} \binom{n-1}{n_b-1} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_k) \\ &= \frac{1}{\binom{n}{n_b}} \frac{1}{n_b} \frac{n_b}{n} \binom{n}{n_b} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_k) \\ &= \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k),\end{aligned}$$

which is the desired result.

ii) This result shows that stochastic gradient methods with a constant step size can only be guaranteed to converge to a neighborhood of the optimal value. It also shows that this neighborhood becomes tighter as n_b grows. *Note that the bound is overly pessimistic when $n_b = n$, since the method is equivalent to gradient descent in that case.*

d) There are several possibilities to guarantee better convergence properties. We cite below the key ones covered in class:

- Use a decreasing step size sequence instead of a constant one;
- Use a gradient aggregation technique (SAGA, SAG, SVRG) instead of a basic stochastic gradient update.

e) If stochastic gradient ($n_b = 1$) improves over gradient descent ($n_b = n$), this indicates there is enough correlation in the data to converge using random subsets of it at every iteration. Using more than one data point yet significantly less than n (mini-batching) can reduce the variance of the gradient estimates while remaining significantly cheaper than a full gradient estimation: this can explain why $n_b = n/10$ yields better performance than $n_b = 1$. When the batch size gets closer to n, its cost also gets closer to that of a full gradient iteration, and the method becomes at risk of suffering from redundancies in the data. This can explain why the behavior of the method worsens when $n_b > n/10$. *Note: This is an open question. Students should be able to a) provide intuition as to why stochastic gradient works better than gradient descent and b) distinguish the mini-batch regime (n_b relatively small) from the regime $n_b \approx n$, where the method tends to behave like gradient descent.*

Solutions of Exercise 2: Importance sampling

a) This sampling strategy will favor components that have larger Lipschitz constants and thus are more likely to vary between successive iterates.

b) Using the distribution of i_k, we have that

$$\begin{aligned}\mathbb{E}_{i_k}\left[\frac{1}{c_{i_k}}\nabla f_{i_k}(\boldsymbol{x}_k)\right] &= \sum_{i=1}^{n}\mathbb{P}\left(i_k=i\right)\frac{1}{c_i}\nabla f_i(\boldsymbol{x}_k)\\ &= \sum_{i=1}^{n}\frac{c_i}{\sum_{j=1}^{n}c_j}\frac{1}{c_i}\nabla f_i(\boldsymbol{x}_k)\\ &= \sum_{i=1}^{n}\frac{1}{\sum_{j=1}^{n}c_j}\nabla f_i(\boldsymbol{x}_k)\\ &= \sum_{i=1}^{n}\frac{1}{n}\nabla f_i(\boldsymbol{x}_k)=\nabla f(\boldsymbol{x}_k),\end{aligned}$$

where we used $\sum_{j=1}^{n}c_j=\sum_{j=1}^{n}\frac{nc_j}{\sum_{\ell=1}^{n}c_\ell}=n$.

c) (Comparison with classical SG).

i) Since $\alpha_k=\frac{1}{L}$, we have

$$\frac{\alpha_k}{c_{i_k}}=\frac{1}{L}\frac{\sum_{j=1}^{n}L_j}{nL_{i_k}}=\frac{n}{\sum_{j=1}^{n}L_j}\frac{\sum_{j=1}^{n}L_j}{nL_{i_k}}=\frac{1}{L_{i_k}}.$$

ii) If i_k is the random index drawn at iteration k, the kth iteration of stochastic gradient reads

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\alpha_k\nabla f_{i_k}(\boldsymbol{x}_k)=\boldsymbol{x}_k-\frac{1}{L}\nabla f_{i_k}(\boldsymbol{x}_k),$$

while the iteration (4.0.6) becomes

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\frac{\alpha_k}{c_{i_k}}\nabla f_{i_k}(\boldsymbol{x}_k)=\boldsymbol{x}_k-\frac{1}{L_i}\nabla f_{i_k}(\boldsymbol{x}_k).$$

As a result, the second iteration will take a smaller stepsize in the direction $-\nabla f_{i_k}(\boldsymbol{x}_k)$ if $L_i\geq\frac{1}{n}\sum_{j=1}^{n}L_j$, i.e. when the ith Lipschitz constant is larger than the average. This is precisely what importance sampling aims at achieving: the stepsize is adjusted according to the Lipschitz constant, in order to reduce the impact of the components with an excessively large Lipschitz constant.

Solutions of Exercise 3: Practical stochastic gradient variants

Consider the k-th iteration of the batch method, and let $\boldsymbol{g}_k=\frac{1}{|S_k|}\sum_{i\in S_k}\nabla f_i(\boldsymbol{x}_k)$. Because the gradient estimates are unbiased by assumptions, we can apply the result of Proposition 2.3.1, which gives:

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1})\right]\leq f(\boldsymbol{x}_k)-\alpha_k\nabla f(\boldsymbol{x}_k)^{\mathrm{T}}\,\mathbb{E}_{i_k}\left[\boldsymbol{g}_k\right]+\frac{L\alpha_k^2}{2}\,\mathbb{E}_{i_k}\left[\|\boldsymbol{g}_k\|^2\right].$$

Since $\boldsymbol{g}_k$ is an unbiased estimate of $\nabla f(\boldsymbol{x}_k)$ by assumption, and given the result of Proposition 3.1.1, we obtain

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] &\leq -\left(\alpha_k - \frac{L\alpha_k^2}{2n_b}\right)\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{L\alpha_k^2}{2n_b}\sigma^2 \\ &\leq -2\mu\left(\alpha_k - \frac{L\alpha_k^2}{2n_b}\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha_k^2}{2n_b}\sigma^2 \\ &= -2\mu\alpha\left(1 - \frac{L\alpha}{2n_b}\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha}{2n_b}\sigma^2.\end{aligned}$$

Using that $\alpha \leq \frac{1}{L}$ then gives

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)\right] \leq -\mu\alpha\left(2 - \frac{1}{n_b}\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2n_b}\sigma^2.$$

Noticing that $\mathbb{E}_{i_k}\left[f^* - f(\boldsymbol{x}_k)\right] = f^* - f(\boldsymbol{x}_k)$, the left-hand side can be modified by adding and subtracting f^*, leading to

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] + f^* - f(\boldsymbol{x}_k) &\leq -\mu\alpha\left(2 - \frac{1}{n_b}\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2n_b}\sigma^2 \\ \mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] &\leq \left(1 - \mu\alpha(2 - \tfrac{1}{n_b})\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2n_b}\sigma^2.\end{aligned}$$

Subtracting $\frac{L\alpha\sigma^2}{2\mu(2n_b-1)}$ on both sides gives

$$\begin{aligned}\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] - \frac{L\alpha}{2\mu(2n_b-1)}\sigma^2 &\leq \left(1 - \mu\alpha(2 - \tfrac{1}{n_b})\right)(f(\boldsymbol{x}_k) - f^*) + \frac{L\alpha^2}{2}\sigma^2 - \frac{L\alpha}{2\mu(2n_b-1)}\sigma^2 \\ &= \left(1 - \mu\alpha(2 - \tfrac{1}{n_b})\right)\left[(f(\boldsymbol{x}_k) - f^*) - \frac{L\alpha}{2\mu(2n_b-1)}\sigma^2\right].\end{aligned}$$

Finally, taking the expected value with respect to every index $i_0, \ldots, i_k$, we arrive at

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) - f^*\right] - \frac{L\alpha}{2\mu(2n_b-1)}\sigma^2 \leq \left(1 - \mu\alpha(2 - \tfrac{1}{n_b})\right)\left[\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] - \frac{L\alpha}{2\mu(2n_b-1)}\sigma^2\right].$$

By iterating over all iterations, we arrive at the desired result.

In order for this result to be meaningful, one must guarantee $1 - \mu\alpha(2 - \frac{1}{n_b}) > 0$ which holds provided $\alpha \leq \frac{n_b}{\mu(2n_b-1)}$. The step size may thus need to be smaller than $\frac{1}{L}$, and a factor of $\frac{n_b}{2n_b-1}$ may have a significant impact in practice.