lecture notes:

Optimization for Machine Learning

version 0.57

Elad Hazan [1]

[1] www.cs.princeton.edu/~ehazan

Preface

This text was written to accompany a series of lectures given at the Machine Learning Summer School Buenos Aires, following a lecture series at the Simons Center for Theoretical Computer Science, Berkeley. It was extended for the course COS 598D - Optimization for Machine Learning, Princeton University, Spring 2019.

I am grateful to Paula Gradu for proofreading parts of this manuscript. I'm also thankful for the help of the following students and colleagues for corrections and suggestions to this text: Udaya Ghai, John Hallman, Noé Pion, Xinyi Chen.

Figure 1: Professor Arkadi Nemirovski, Pioneer of mathematical optimization

Contents

Notation

We use the following mathematical notation in this writeup:

- d-dimensional Euclidean space is denoted $\mathbb{R}^d$.
- Vectors are denoted by boldface lower-case letters such as $\mathbf{x} \in \mathbb{R}^d$. Coordinates of vectors are denoted by underscore notation $\mathbf{x}_i$ or regular brackets $\mathbf{x}(i)$.
- Matrices are denoted by boldface upper-case letters such as $\mathbf{X} \in \mathbb{R}^{m \times n}$. Their coordinates by $\mathbf{X}(i, j)$, or $\mathbb{X}_{ij}$.
- Functions are denoted by lower case letters $f : \mathbb{R}^d \mapsto \mathbb{R}$.
- The k-th differential of function f is denoted by $\nabla^k f \in \mathbb{R}^{d^k}$. The gradient is denoted without the superscript, as ∇f.
- We use the mathcal macro for sets, such as $\mathcal{K} \subseteq \mathbb{R}^d$.
- We denote the gradient at point $\mathbf{x}_t$ as $\nabla_{\mathbf{x}_t}$, or simply ∇_t.
- We denote the global or local optima of functions by $\mathbf{x}^\star$.
- We denote distance to optimality for iterative algorithms by $h_t = f(\mathbf{x}_t) - f(\mathbf{x}^\star)$.
- Euclidean distance to optimality is denoted $d_t = \|\mathbf{x}_t - \mathbf{x}^\star\|$.

Chapter 1

Introduction

The topic of this lecture series is the mathematical optimization approach to machine learning.

In standard algorithmic theory, the burden of designing an efficient algorithm for solving a problem at hand is on the algorithm designer. In the decades since in the introduction of computer science, elegant algorithms have been designed for tasks ranging from finding the shortest path in a graph, computing the optimal flow in a network, compressing a computer file containing an image captured by digital camera, and replacing a string in a text document.

The design approach, while useful to many tasks, falls short of more complicated problems, such as identifying a particular person in an image in bitmap format, or translating text from English to Hebrew. There may very well be an elegant algorithm for the above tasks, but the algorithmic design scheme does not scale.

As Turing promotes in his paper [83], it is potentially easier to teach a computer to learn how to solve a task, rather than teaching it the solution for the particular tasks. In effect, that's what we do at school, or in this lecture series...

The machine learning approach to solving problems is to have an automated mechanism for learning an algorithm. Consider the problem of classifying images into two categories: those containing cars and those containing chairs (assuming there are only two types of images in the world). In ML we train (teach) a machine to achieve the desired functionality. The same machine can potentially solve any algorithmic task, and differs from task to task only by a set of parameters that determine the functionality of the machine. This is much like the wires in a computer chip determine its

functionality. Indeed, one of the most popular machines are artificial neural networks.

The mathematical optimization approach to machine learning is to view the process of machine training as an optimization problem. If we let $w \in \mathbb{R}^d$ be the parameters of our machine (a.k.a. model), that are constrained to be in some set $\mathcal{K} \subseteq \mathbb{R}^d$, and f the function measuring success in mapping examples to their correct label, then the problem we are interested in is described by the mathematical optimization problem of

$$\boxed{\min_{w \in \mathcal{K}} f(w)} \tag{1.1}$$

This is the problem that the lecture series focuses on, with particular emphasis on functions that arise in machine learning and have special structure that allows for efficient algorithms.

1.1 Examples of optimization problems in machine learning

1.1.1 Empirical Risk Minimization

Machine learning problems exhibit special structure. For example, one of the most basic optimization problems in supervised learning is that of fitting a model to data, or examples, also known as the optimization problem of Empirical Risk Minimization (ERM). The special structure of the problems arising in such formulations is separability across different examples into individual losses.

An example of such formulation is the supervised learning paradigm of linear classification. In this model, the learner is presented with positive and negative examples of a concept. Each example, denoted by $\mathbf{a}_i$, is represented in Euclidean space by a d dimensional feature vector. For example, a common representation for emails in the spam-classification problem are binary vectors in Euclidean space, where the dimension of the space is the number of words in the language. The i'th email is a vector $\mathbf{a}_i$ whose entries are given as ones for coordinates corresponding to words that appear in the email, and zero otherwise[1]. In addition, each example has a label $b_i \in \{-1, +1\}$, corresponding to whether the email has been labeled spam/not spam. The

[1]Such a representation may seem naïve at first as it completely ignores the words' order of appearance and their context. Extensions to capture these features are indeed studied in the Natural Language Processing literature.

goal is to find a hyperplane separating the two classes of vectors: those with positive labels and those with negative labels. If such a hyperplane, which completely separates the training set according to the labels, does not exist, then the goal is to find a hyperplane that achieves a separation of the training set with the smallest number of mistakes.

Mathematically speaking, given a set of m examples to train on, we seek $\mathbf{x} \in \mathbb{R}^d$ that minimizes the number of incorrectly classified examples, i.e.

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{m} \sum_{i\in[m]} \delta(\text{sign}(\mathbf{x}^\top \mathbf{a}_i) \neq b_i) \tag{1.2}$$

where $\text{sign}(x) \in \{-1, +1\}$ is the sign function, and $\delta(z) \in \{0, 1\}$ is the indicator function that takes the value 1 if the condition z is satisfied and zero otherwise.

The mathematical formulation of the linear classification above is a special case of mathematical programming (1.1), in which

$$f(\mathbf{x}) = \frac{1}{m} \sum_{i\in[m]} \delta(\text{sign}(\mathbf{x}^\top \mathbf{a}_i) \neq b_i) = \mathop{\mathbf{E}}_{i\sim[m]}[\ell_i(\mathbf{x})],$$

where we make use of the expectation operator for simplicity, and denote $\ell_i(\mathbf{x}) = \delta(\text{sign}(\mathbf{x}^\top \mathbf{a}_i) \neq b_i)$ for brevity. Since the program above is non-convex and non-smooth, it is common to take a convex relaxation and replace ℓ_i with convex loss functions. Typical choices include the means square error function and the hinge loss, given by

$$\ell_{\mathbf{a}_i,b_i}(\mathbf{x}) = \max\{0, 1 - b_i \cdot \mathbf{x}^\top \mathbf{a}_i\}.$$

This latter loss function in the context of binary classification gives rise to the popular soft-margin SVM problem.

Another important optimization problem is that of training a deep neural network for binary classification. For example, consider a dataset of images, represented in bitmap format and denoted by $\{\mathbf{a}_i \in \mathbb{R}^d | i \in [m]\}$, i.e. m images over n pixels. We would like to find a mapping from images to the two categories, $\{b_i \in \{0, 1\}\}$ of cars and chairs. The mapping is given by a set of parameters of a machine class, such as weights in a neural network, or values of a support vector machine. We thus try to find the optimal parameters that match $\mathbf{a}_i$ to b, i..e

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) = \mathop{\mathbf{E}}_{\mathbf{a}_i,b_i} \left[\ell(f_{\mathbf{w}}(\mathbf{a}_i), b_i)\right].$$

1.1.2 Matrix completion and recommender systems

Media recommendations have changed significantly with the advent of the Internet and rise of online media stores. The large amounts of data collected allow for efficient clustering and accurate prediction of users' preferences for a variety of media. A well-known example is the so called "Netflix challenge"—a competition of automated tools for recommendation from a large dataset of users' motion picture preferences.

One of the most successful approaches for automated recommendation systems, as proven in the Netflix competition, is matrix completion. Perhaps the simplest version of the problem can be described as follows.

The entire dataset of user-media preference pairs is thought of as a partially-observed matrix. Thus, every person is represented by a row in the matrix, and every column represents a media item (movie). For simplicity, let us think of the observations as binary—a person either likes or dislikes a particular movie. Thus, we have a matrix $M \in \{0, 1, *\}^{n \times m}$ where n is the number of persons considered, m is the number of movies at our library, and $0/1$ and $*$ signify "dislike", "like" and "unknown" respectively:

$$M_{ij} = \begin{cases} 0, & \text{person } i \text{ dislikes movie } j \\ 1, & \text{person } i \text{ likes movie } j \\ *, & \text{preference unknown} \end{cases} .$$

The natural goal is to complete the matrix, i.e. correctly assign 0 or 1 to the unknown entries. As defined so far, the problem is ill-posed, since any completion would be equally good (or bad), and no restrictions have been placed on the completions.

The common restriction on completions is that the "true" matrix has low rank. Recall that if a matrix $X \in \mathbb{R}^{n \times m}$ has rank $k \leq \rho = \min\{n, m\}$ then it can be written as

$$X = UV \ , \ U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{k \times m}.$$

The intuitive interpretation of this property is that each entry in M can be explained by only k numbers. In matrix completion this means, intuitively, that there are only k factors that determine a persons preference over movies, such as genre, director, actors and so on.

Now the simplistic matrix completion problem can be well-formulated as in the following mathematical program. Denote by $\|\cdot\|_{OB}$ the Euclidean

norm only on the observed (non starred) entries of M, i.e.,

$$\|X\|_{OB}^2 = \sum_{M_{ij} \neq *} X_{ij}^2.$$

The mathematical program for matrix completion is given by

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times m}} \ & \frac{1}{2}\|X - M\|_{OB}^2 \\ \text{s.t.} \quad & \text{rank}(X) \leq k. \end{aligned}$$

1.1.3 Learning in Linear Dynamical Systems

Many learning problems require memory, or the notion of state. This is captured by the paradigm of reinforcement learning, as well of the special case of control in Linear Dynamical Systems (LDS).

LDS model a variety of control and robotics problems in continuous variables. The setting is that of a time series, with following parameters:

1. Inputs to the system, also called controls, denoted by $\mathbf{u}_1, ..., \mathbf{u}_T \in \mathbb{R}^n$.

2. Outputs from the system, also called observations, denoted $\mathbf{y}_1, ..., \mathbf{y}_T \in \mathbb{R}^m$.

3. The state of the system, which may either be observed or hidden, denoted $\mathbf{x}_t, ..., \mathbf{x}_T \in \mathbb{R}^d$.

4. The system parameters, which are transformations matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ in appropriate dimensions.

In the online learning problem of LDS, the learner iteratively observes $\mathbf{u}_t, \mathbf{y}_t$, and has to predict $\hat{\mathbf{y}}_{t+1}$. The actual $\mathbf{y}_t$ is generated according to the following dynamical equations:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \varepsilon_t \\ \mathbf{y}_{t+1} &= \mathbf{C}\mathbf{x}_{t+1} + \mathbf{D}\mathbf{u}_t + \zeta_t, \end{aligned}$$

where ε_t, ζ_t are noise which is distributed as a Normal random variable.

Consider an online sequence in which the states are visible. At time t, all system states, inputs and outputs are visible up to this time step. The learner has to predict $\mathbf{y}_{t+1}$, and only afterwards observes $\mathbf{u}_{t+1}.\mathbf{x}_{t+1}, \mathbf{y}_{t+1}$.

One reasonable way to predict $\mathbf{y}_{t+1}$ based upon past observations is to compute the system, and use the computed transformations to predict. This amounts to solving the following mathematical program:

$$\min_{\mathbf{A},\mathbf{B},\hat{\mathbf{C}},\hat{\mathbf{D}}} \left\{ \sum_{\tau<t} (\mathbf{x}_{\tau+1} - \mathbf{A}\mathbf{x}_\tau + \mathbf{B}\mathbf{u}_\tau)^2 + (\mathbf{y}_{\tau+1} - \hat{\mathbf{C}}\mathbf{x}_\tau + \hat{\mathbf{D}}\mathbf{u}_\tau)^2 \right\},$$

and then predicting $\hat{\mathbf{y}}_{t+1} = \hat{\mathbf{C}}\hat{\mathbf{A}}(\mathbf{x}_t + \mathbf{B}\mathbf{u}_t) + \hat{\mathbf{D}}\mathbf{u}_t$.

1.2 Why is mathematical programming hard?

The general formulation (1.1) is NP hard. To be more precise, we have to define the computational model we are working in as well as and the access model to the function.

Before we give a formal proof, the intuition to what makes mathematical optimization hard is simple to state. In one line: it is the fact that global optimality cannot be verified on the basis of local properties.

Most, if not all, efficient optimization algorithms are iterative and based on a local improvement step. By this nature, any optimization algorithm will terminate when the local improvement is no longer possible, giving rise to a proposed solution. However, the quality of this proposed solution may differ significantly, in general, from that of the global optimum.

This intuition explains the need for a property of objectives for which global optimality is locally verifiable. Indeed, this is exactly the notion of **convexity**, and the reasoning above explains its utmost importance in mathematical optimization.

We now to prove that mathematical programming is NP-hard. This requires discussion of the computational model as well as access model to the input.

1.2.1 The computational model

The computational model we shall adopt throughout this manuscript is that of a RAM machine equipped with oracle access to the objective function $f : \mathbb{R}^d \mapsto \mathbb{R}$ and constraints set $\mathcal{K} \subseteq \mathbb{R}^d$. The oracle model for the objective function can be one of the following, depending on the specific scenario:

1. **Value oracle:** given a point $x \in \mathbb{R}^d$, oracle returns $f(x) \in \mathbb{R}$.
2. **Gradient (first-order) oracle:** given a point $x \in \mathbb{R}^d$, oracle returns the gradient $\nabla f(x) \in \mathbb{R}^d$.

3. **k-th order differential oracle:** given a point $x \in \mathbb{R}^d$, oracle returns the tensor $\nabla^k f(x) \in \mathbb{R}^{d^k}$.

The oracle model for the constraints set is a bit more subtle. We distinguish between the following oracles:

1. **Membership oracle:** given a point $x \in \mathbb{R}^d$, oracle returns one if $x \in \mathcal{K}$ and zero otherwise.

2. **Separating hyperplane oracle:** given a point $x \in \mathbb{R}^d$, oracle either returns "Yes" if $x \in \mathcal{K}$, or otherwise returns a hyperplane $h \in \mathbb{R}^d$ such that $h^\top x > 0$ and $\forall y \in \mathcal{K}$, $h^\top y \leq 0$.

3. **Explicit sets:** the most common scenario in machine learning is one in which $\mathcal{K}$ is "natural", such as the Euclidean ball or hypercube, or the entire Euclidean space.

1.2.2 Hardness of constrained mathematical programming

Under this computational model, we can show:

Lemma 1.1. *Mathematical programming is NP-hard, even for a convex continuous constraint set $\mathcal{K}$ and quadratic objective functions.*

Informal sketch. Consider the MAX-CUT problem: given a graph $G = (V, E)$, find a subset of the vertices that maximizes the number of edges cut. Let A be the negative adjacency matrix of the graph, i.e.

$$A_{ij} = \begin{cases} -1, & (i,j) \in E \\ \\ 0, & o/w \end{cases}$$

Also suppose that $A_{ii} = 0$.

Next, consider the mathematical program:

$$\min \left\{ f_A(\mathbf{x}) = \frac{1}{4}(\mathbf{x}^\top A \mathbf{x} - 2|E|) \right\} \qquad (1.3)$$
$$\|\mathbf{x}\|_\infty = 1 \ .$$

Consider the cut defined by the solution of this program, namely

$$S_{\mathbf{x}} = \{i \in V | \mathbf{x}_i = 1\},$$

for $\mathbf{x} = \mathbf{x}^\star$. Let $C(S)$ denote the size of the cut specified by the subset of edges $S \subseteq E$. Observe that the expression $\frac{1}{2}\mathbf{x}^\top A \mathbf{x}$, is exactly equal to the

number of edges that are cut by $S_{\mathbf{x}}$ minus the number of edges that are uncut. Thus, we have

$$\frac{1}{2}\mathbf{x}A\mathbf{x} = C(S_{\mathbf{x}}) - (E - C(S_{\mathbf{x}})) = 2C(S_{\mathbf{x}}) - E,$$

and hence $f(\mathbf{x}) = C(S_{\mathbf{x}})$. Therefore, maximizing $f(\mathbf{x})$ is equivalent to the MAX-CUT problem, and is thus NP-hard. We proceed to make the constraint set convex and continuous. Consider the mathematical program

$$\begin{aligned} &\min\{f_A(\mathbf{x})\} \\ &\|\mathbf{x}\|_\infty \leq 1 \ . \end{aligned} \tag{1.4}$$

This is very similar to the previous program, but we relaxed the equality to be an inequality, consequently the constraint set is now the hypercube. We now claim that the solution is w.l.o.g. a vertex. To see that, consider $\mathbf{y}(\mathbf{x}) \in \{\pm 1\}^d$ a rounding of $\mathbf{x}$ to the corners defined by:

$$\mathbf{y}_i = \mathbf{y}(\mathbf{x})_i = \begin{cases} 1, & w.p. \frac{1+\mathbf{x}_i}{2} \\ -1, & w.p. \frac{1-\mathbf{x}_i}{2} \end{cases}$$

Notice that

$$\mathbf{E}[\mathbf{y}] = \mathbf{x} \ , \ \forall i \neq j \ . \ \mathbf{E}[\mathbf{y}_i\mathbf{y}_j] = \mathbf{x}_i\mathbf{x}_j,$$

and therefore $\mathbf{E}[\mathbf{y}(\mathbf{x})^\top A\mathbf{y}(\mathbf{x})] = \mathbf{x}^\top A\mathbf{x}$. We conclude that the optimum of mathematical program 1.4 is the same as that for 1.3, and both are NP-hard. □

Chapter 2

Basic concepts in optimization and analysis

2.1 Basic definitions and the notion of convexity

We consider minimization of a continuous function over a convex subset of Euclidean space. We mostly consider objective functions that are convex. In later chapters we relax this requirement and consider non-convex functions as well.

Henceforth, let $\mathcal{K} \subseteq \mathbb{R}^d$ be a bounded convex and compact set in Euclidean space. We denote by D an upper bound on the diameter of $\mathcal{K}$:

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{K},\ \|\mathbf{x} - \mathbf{y}\| \leq D.$$

A set $\mathcal{K}$ is convex if for any $\mathbf{x}, \mathbf{y} \in \mathcal{K}$, all the points on the line segment connecting $\mathbf{x}$ and $\mathbf{y}$ also belong to $\mathcal{K}$, i.e.,

$$\forall \alpha \in [0,1],\ \alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathcal{K}.$$

A function $f : \mathcal{K} \mapsto \mathbb{R}$ is convex if for any $\mathbf{x}, \mathbf{y} \in \mathcal{K}$

$$\forall \alpha \in [0,1],\ f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}).$$

Gradients and subgradients. The set of all subgradients of a function f at $\mathbf{x}$, denoted $\partial f(\mathbf{x})$, is the set of all vectors $\mathbf{u}$ such that

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{u}^\top(\mathbf{y} - \mathbf{x}).$$

It can be shown that the set of subgradients of a convex function is always non-empty.

Suppose f is differentiable, let $\nabla f(\mathbf{x})[i] = \frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x})$ be the vector of partial derivatives according to the variables, called the gradient. If the gradient $\nabla f(\mathbf{x})$ exists, then $\nabla f(\mathbf{x}) \in \partial f(\mathbf{x})$ and $\forall \mathbf{y} \in \mathcal{K}$

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}).$$

Henceforth we shall denote by $\nabla f(\mathbf{x})$ the gradient, if it exists, or any member of $\partial f(\mathbf{x})$ otherwise.

We denote by $G > 0$ an upper bound on the norm of the subgradients of f over $\mathcal{K}$, i.e., $\|\nabla f(\mathbf{x})\| \leq G$ for all $\mathbf{x} \in \mathcal{K}$. The existence of Such an upper bound implies that the function f is Lipschitz continuous with parameter G, that is, for all $\mathbf{x}, \mathbf{y} \in \mathcal{K}$

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq G\|\mathbf{x} - \mathbf{y}\|.$$

Smoothness and strong convexity. The optimization and machine learning literature studies special types of convex functions that admit useful properties, which in turn allow for more efficient optimization. Notably, we say that a function is α-strongly convex if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{\alpha}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

A function is β-smooth if

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{\beta}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

The latter condition is implied by a slightly stronger Lipschitz condition over the gradients, which is sometimes used to defined smoothness, i.e.,

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta\|\mathbf{x} - \mathbf{y}\|.$$

If the function is twice differentiable and admits a second derivative, known as a Hessian for a function of several variables, the above conditions are equivalent to the following condition on the Hessian, denoted $\nabla^2 f(\mathbf{x})$:

$$\begin{aligned} \text{Smoothness:} \quad & -\beta I \preccurlyeq \nabla^2 f(\mathbf{x}) \preccurlyeq \beta I \\ \text{Strong-convexity:} \quad & \alpha I \preccurlyeq \nabla^2 f(\mathbf{x}), \end{aligned}$$

where $A \preccurlyeq B$ if the matrix $B - A$ is positive semidefinite.

When the function f is both α-strongly convex and β-smooth, we say that it is γ-well-conditioned where γ is the ratio between strong convexity and smoothness, also called the *condition number* of f

$$\gamma = \frac{\alpha}{\beta} \leq 1$$

2.1.1 Projections onto convex sets

In the following algorithms we shall make use of a projection operation onto a convex set, which is defined as the closest point inside the convex set to a given point. Formally,

$$\underset{\mathcal{K}}{\Pi}(\mathbf{y}) \triangleq \arg\min_{\mathbf{x}\in\mathcal{K}} \|\mathbf{x}-\mathbf{y}\|.$$

When clear from the context, we shall remove the $\mathcal{K}$ subscript. It is left as an exercise to the reader to prove that the projection of a given point over a closed non-empty convex set exists and is unique.

The computational complexity of projections is a subtle issue that depends much on the characterization of $\mathcal{K}$ itself. Most generally, $\mathcal{K}$ can be represented by a membership oracle—an efficient procedure that is capable of deciding whether a given $\mathbf{x}$ belongs to $\mathcal{K}$ or not. In this case, projections can be computed in polynomial time. In certain special cases, projections can be computed very efficiently in near-linear time.

A crucial property of projections that we shall make extensive use of is the Pythagorean theorem, which we state here for completeness:

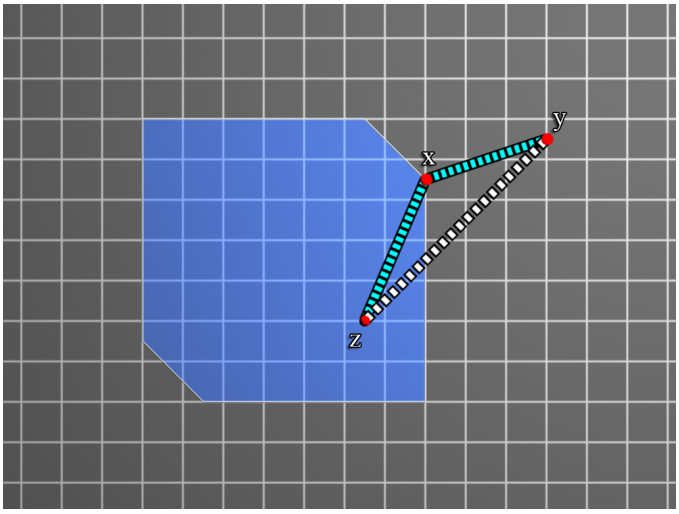

Figure 2.1: Pythagorean theorem.

Theorem 2.1 (Pythagoras, circa 500 BC)**.** *Let $\mathcal{K} \subseteq \mathbb{R}^d$ be a convex set, $\mathbf{y} \in \mathbb{R}^d$ and $\mathbf{x} = \Pi_{\mathcal{K}}(\mathbf{y})$. Then for any $\mathbf{z} \in \mathcal{K}$ we have*

$$\|\mathbf{y}-\mathbf{z}\| \geq \|\mathbf{x}-\mathbf{z}\|.$$

We note that there exists a more general version of the Pythagorean theorem. The above theorem and the definition of projections are true and valid not only for Euclidean norms, but for projections according to other distances that are not norms. In particular, an analogue of the Pythagorean theorem remains valid with respect to Bregman divergences.

2.1.2 Introduction to optimality conditions

The standard curriculum of high school mathematics contains the basic facts concerning when a function (usually in one dimension) attains a local optimum or saddle point. The KKT (Karush-Kuhn-Tucker) conditions generalize these facts to more than one dimension, and the reader is referred to the bibliographic material at the end of this chapter for an in-depth rigorous discussion of optimality conditions in general mathematical programming.

For our purposes, we describe only briefly and intuitively the main facts that we will require henceforth. We separate the discussion into convex and non-convex programming.

Optimality for convex optimization

A local minimum of a convex function is also a global minimum (see exercises at the end of this chapter). We say that $\mathbf{x}^\star$ is an ε-approximate optimum if the following holds:

$$\forall \mathbf{x} \in \mathcal{K} \ . \ f(\mathbf{x}^\star) \leq f(\mathbf{x}) + \varepsilon.$$

The generalization of the fact that a minimum of a convex differentiable function on $\mathbb{R}$ is a point in which its derivative is equal to zero, is given by the multi-dimensional analogue that its gradient is zero:

$$\nabla f(\mathbf{x}) = 0 \quad \Longleftrightarrow \quad \mathbf{x} \in \arg\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}).$$

We will require a slightly more general, but equally intuitive, fact for constrained optimization: at a minimum point of a constrained convex function, the inner product between the negative gradient and direction towards the interior of $\mathcal{K}$ is non-positive. This is depicted in Figure 2.2, which shows that $-\nabla f(\mathbf{x}^\star)$ defines a supporting hyperplane to $\mathcal{K}$. The intuition is that if the inner product were positive, one could improve the objective by moving in the direction of the projected negative gradient. This fact is stated formally in the following theorem.

Theorem 2.2 (Karush-Kuhn-Tucker)**.** *Let $\mathcal{K} \subseteq \mathbb{R}^d$ be a convex set, $\mathbf{x}^\star \in \arg\min_{\mathbf{x}\in\mathcal{K}} f(\mathbf{x})$. Then for any $\mathbf{y} \in \mathcal{K}$ we have*

$$\nabla f(\mathbf{x}^\star)^\top (\mathbf{y} - \mathbf{x}^\star) \geq 0.$$

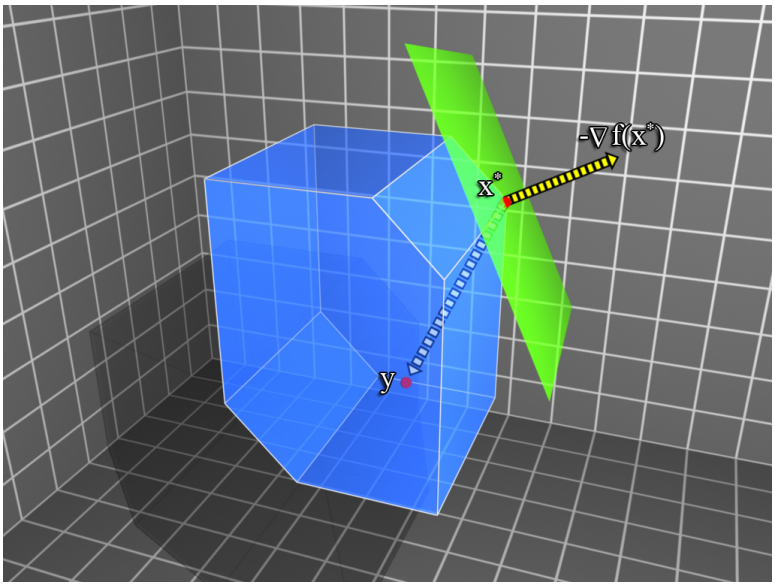

Figure 2.2: Optimality conditions: negative (sub)gradient pointing outwards.

2.1.3 Solution concepts for non-convex optimization

We have seen in the previous chapter that mathematical optimization is NP-hard. This implies that finding global solutions for non-convex optimization is NP-hard, even for smooth functions over very simple convex domains. We thus consider other trackable concepts of solutions.

The most common solution concept is that of first-order optimality, a.k.a. saddle-points or stationary points. These are points that satisfy

$$\|\nabla f(\mathbf{x}^\star)\| = 0.$$

Unfortunately, even finding such stationary points is NP-hard. We thus settle for approximate stationary points, which satisify

$$\|\nabla f(\mathbf{x}^\star)\| \leq \varepsilon.$$

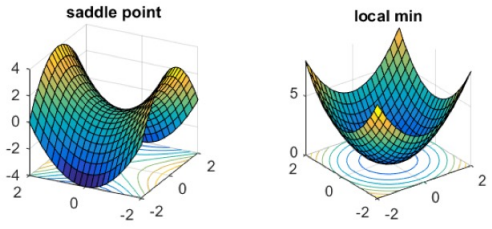

Figure 2.3: First and second-order local optima.

A more stringent notion of optimality we may consider is obtained by looking at the second derivatives. We can require they behave as for global minimum, see figure 2.3. Formally, we say that a point $\mathbf{x}^\star$ is a second-order local minimum if it satisfies the two conditions:

$$\|\nabla f(\mathbf{x}^\star)\| \leq \varepsilon \ , \ \nabla^2 f(\mathbf{x}^\star) \succeq -\sqrt{\varepsilon} I.$$

The differences in approximation criteria for first and second derivatives is natural, as we shall explore in non-convex approximation algorithms henceforth.

We note that it is possible to further define optimality conditions for higher order derivatives, although this is less useful in the context of machine learning.

2.2 Potentials for distance to optimality

When analyzing convergence of gradient methods, it is useful to use potential functions in lieu of function distance to optimality, such as gradient norm and/or Euclidean distance to optimality. The following relationships hold between these quantities.

Lemma 2.3. *The following properties hold for α-strongly-convex functions and/or β-smooth functions over Euclidean space $\mathbb{R}^d$.*

1. $\frac{\alpha}{2} d_t^2 \leq h_t$
2. $h_t \leq \frac{\beta}{2} d_t^2$

3. $\frac{1}{2\beta}\|\nabla_t\|^2 \le h_t$
4. $h_t \le \frac{1}{2\alpha}\|\nabla_t\|^2$

Proof. 1. $h_t \ge \frac{\alpha}{2} d_t^2$:

By strong convexity, we have

$$\begin{aligned} h_t &= f(\mathbf{x}_t) - f(\mathbf{x}^\star) \\ &\ge \nabla f(\mathbf{x}^\star)^\top (\mathbf{x}_t - \mathbf{x}^\star) + \tfrac{\alpha}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|^2 \\ &= \tfrac{\alpha}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|^2 \end{aligned}$$

where the last inequality follows since the gradient at the global optimum is zero.

2. $h_t \le \frac{\beta}{2} d_t^2$:

By smoothness,

$$\begin{aligned} h_t &= f(\mathbf{x}_t) - f(\mathbf{x}^\star) \\ &\le \nabla f(\mathbf{x}^\star)^\top (\mathbf{x}_t - \mathbf{x}^\star) + \tfrac{\beta}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|^2 \\ &= \tfrac{\beta}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|^2 \end{aligned}$$

where the last inequality follows since the gradient at the global optimum is zero.

3. $h_t \ge \frac{1}{2\beta}\|\nabla_t\|^2$: Using smoothness, and let $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\nabla_t$ for $\eta = \frac{1}{\beta}$,

$$\begin{aligned} h_t = \; & f(\mathbf{x}_t) - f(\mathbf{x}^\star) \\ & \ge f(\mathbf{x}_t) - f(\mathbf{x}_{t+1}) \\ & \ge \nabla f(\mathbf{x}_t)^\top (\mathbf{x}_t - \mathbf{x}_{t+1}) - \tfrac{\beta}{2}\|\mathbf{x}_t - \mathbf{x}_{t+1}\|^2 \\ & = \eta\|\nabla_t\|^2 - \tfrac{\beta}{2}\eta^2\|\nabla_t\|^2 \\ & = \tfrac{1}{2\beta}\|\nabla_t\|^2. \end{aligned}$$

4. $h_t \le \frac{1}{2\alpha}\|\nabla_t\|^2$:

We have for any pair $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

$$\begin{aligned} f(\mathbf{y}) &\ge f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{\alpha}{2}\|\mathbf{x} - \mathbf{y}\|^2 \\ &\ge \min_{\mathbf{z}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{z} - \mathbf{x}) + \frac{\alpha}{2}\|\mathbf{x} - \mathbf{z}\|^2 \right\} \\ &= f(\mathbf{x}) - \frac{1}{2\alpha}\|\nabla f(\mathbf{x})\|^2. \\ &\text{by taking } \mathbf{z} = \mathbf{x} - \frac{1}{\alpha}\nabla f(\mathbf{x}) \end{aligned}$$

In particular, taking $\mathbf{x} = \mathbf{x}_t$, $\mathbf{y} = \mathbf{x}^\star$, we get

$$h_t = f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \frac{1}{2\alpha}\|\nabla_t\|^2. \tag{2.1}$$

□

2.3 Gradient descent and the Polyak stepsize

The simplest iterative optimization algorithm is gradient descent, as given in Algorithm 1. We analyze GD with the Polyak stepsize, which has the advantage of not depending on the strong convexity and/or smoothness parameters of the objective function.

Algorithm 1 GD with the Polyak stepsize

1: Input: time horizon T, x_0
2: **for** $t = 0, \ldots, T-1$ **do**
3: Set $\eta_t = \frac{h_t}{\|\nabla_t\|^2}$
4: $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla_t$
5: **end for**
6: Return $\bar{\mathbf{x}} = \arg\min_{\mathbf{x}_t}\{f(\mathbf{x}_t)\}$

To prove convergence bounds, assume $\|\nabla_t\| \leq G$, and define:

$$B_T \quad = \quad \min\left\{\frac{Gd_0}{\sqrt{T}}, \frac{2\beta d_0^2}{T}, \frac{3G^2}{\alpha T}, \beta d_0^2\left(1 - \frac{\alpha}{4\beta}\right)^T\right\}$$

Theorem 2.4. *(GD with the Polyak Step Size) Algorithm 1 attains the following regret bound after T steps:*

$$h(\bar{\mathbf{x}}) \quad = \quad \min_{0\leq t\leq T}\{h_t\} \leq B_T$$

Theorem 2.4 directly follows from the following lemma. Let $0 \leq \gamma \leq 1$, define $R_{T,\gamma}$ as follows:

$$R_{T,\gamma} = \min\left\{\frac{Gd_0}{\sqrt{\gamma T}}, \frac{2\beta d_0^2}{\gamma T}, \frac{3G^2}{\gamma\alpha T}, \beta d_0^2\left(1 - \gamma\frac{\alpha}{4\beta}\right)^T\right\}.$$

Lemma 2.5. *For $0 \leq \gamma \leq 1$, suppose that a sequence $\mathbf{x}_0, \ldots \mathbf{x}_t$ satisfies:*

$$d_{t+1}^2 \leq d_t^2 - \gamma\frac{h_t^2}{\|\nabla_t\|^2} \tag{2.2}$$

then for $\bar{\mathbf{x}}$ *as defined in the algorithm, we have:*

$$h(\bar{\mathbf{x}}) \leq R_{T,\gamma}\,.$$

Proof. The proof analyzes different cases:

1. For convex functions with gradient bounded by G,

$$d_{t+1}^2 - d_t^2 \;\; \leq -\tfrac{\gamma h_t^2}{\|\nabla_t\|^2} \leq -\tfrac{\gamma h_t^2}{G^2}$$

   Summing up over T iterations, and using Cauchy-Schwartz, we have

$$\begin{aligned}\frac{1}{T}\sum_t h_t &\leq \frac{1}{\sqrt{T}}\sqrt{\sum_t h_t^2} \\ &\leq \frac{G}{\sqrt{\gamma T}}\sqrt{\sum_t (d_t^2 - d_{t+1}^2)} \leq \frac{G d_0}{\sqrt{\gamma T}}\,.\end{aligned}$$

2. For smooth functions whose gradient is bounded by G, Lemma 2.3 implies:

$$d_{t+1}^2 - d_t^2 \leq -\frac{\gamma h_t^2}{\|\nabla_t\|^2} \leq -\frac{\gamma h_t}{2\beta}\,.$$

   This implies

$$\frac{1}{T}\sum_t h_t \leq \frac{2\beta d_0^2}{\gamma T}\,.$$

3. For strongly convex functions, Lemma 2.3 implies:

$$d_{t+1}^2 - d_t^2 \leq -\gamma\frac{h_t^2}{\|\nabla_t\|^2} \leq -\gamma\frac{h_t^2}{G^2} \leq -\gamma\frac{\alpha^2 d_t^4}{4G^2}\,.$$

   In other words, $d_{t+1}^2 \leq d_t^2(1 - \gamma\frac{\alpha^2 d_t^2}{4G^2})$. Defining $a_t := \gamma\frac{\alpha^2 d_t^2}{4G^2}$, we have:

$$a_{t+1} \leq a_t(1 - a_t)\,.$$

This implies that $a_t \leq \frac{1}{t+1}$, which can be seen by induction[1]. The proof is completed as follows[2] :

$$\begin{aligned}
\frac{1}{T/2}\sum_{t=T/2}^{T} h_t^2 &\leq \frac{2G^2}{\gamma T}\sum_{t=T/2}^{T}(d_t^2 - d_{t+1}^2) \\
&= \frac{2G^2}{\gamma T}(d_{T/2}^2 - d_T^2) \\
&= \frac{8G^4}{\gamma^2\alpha^2 T}(a_{T/2} - a_T) \\
&\leq \frac{9G^4}{\gamma^2\alpha^2 T^2}\,.
\end{aligned}$$

Thus, there exists a t for which $h_t^2 \leq \frac{9G^4}{\gamma^2\alpha^2T^2}$. Taking the square root completes the claim.

4. For both strongly convex and smooth functions:

$$d_{t+1}^2 - d_t^2 \leq -\gamma\frac{h_t^2}{\|\nabla_t\|^2} \leq -\frac{\gamma h_t}{2\beta} \leq -\gamma\frac{\alpha}{4\beta}d_t^2$$

Thus,

$$h_T \leq \beta d_T^2 \leq \beta d_0^2\left(1 - \gamma\frac{\alpha}{4\beta}\right)^T\,.$$

This completes the proof of all cases. □

[1] That $a_0 \leq 1$ follows from Lemma 2.3. For $t = 1$, $a_1 \leq \frac{1}{2}$ since $a_1 \leq a_0(1 - a_0)$ and $0 \leq a_0 \leq 1$. For the induction step, $a_t \leq a_{t-1}(1 - a_{t-1}) \leq \frac{1}{t}(1 - \frac{1}{t}) = \frac{t-1}{t^2} = \frac{1}{t+1}(\frac{t^2-1}{t^2}) \leq \frac{1}{t+1}$.

[2] This assumes T is even. T odd leads to the same constants.

2.4 Exercises

1. Write an explicit expression for the gradient and projection operation (if needed) for each of the example optimization problems in the first chapter.

2. Prove that a differentiable function $f(x) : \mathbb{R} \to \mathbb{R}$ is convex if and only if for any $x, y \in \mathbb{R}$ it holds that $f(x) - f(y) \leq (x - y)f'(x)$.

3. Recall that we say that a function $f : \mathbb{R}^n \to \mathbb{R}$ has a condition number $\gamma = \alpha/\beta$ over $K \subseteq \mathbb{R}^d$ if the following two inequalities hold for all $\mathbf{x}, \mathbf{y} \in \mathcal{K}$:

   (a) $f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\top \nabla f(\mathbf{x}) + \frac{\alpha}{2}\|\mathbf{x} - \mathbf{y}\|^2$

   (b) $f(\mathbf{y}) \leq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\top \nabla f(\mathbf{x}) + \frac{\beta}{2}\|\mathbf{x} - \mathbf{y}\|^2$

   For matrices $A, B \in \mathbb{R}^{n \times n}$ we denote $A \succcurlyeq B$ if $A - B$ is positive semidefinite. Prove that if f is twice differentiable and it holds that $\beta\mathbf{I} \succcurlyeq \nabla^2 f(\mathbf{x}) \succcurlyeq \alpha\mathbf{I}$ for any $\mathbf{x} \in \mathcal{K}$, then the condition number of f over $\mathcal{K}$ is α/β.

4. Prove:

   (a) The sum of convex functions is convex.

   (b) Let f be α_1-strongly convex and g be α_2-strongly convex. Then $f + g$ is $(\alpha_1 + \alpha_2)$-strongly convex.

   (c) Let f be β_1-smooth and g be β_2-smooth. Then $f + g$ is $(\beta_1 + \beta_2)$-smooth.

5. Let $\mathcal{K} \subseteq \mathbb{R}^d$ be closed, compact, non-empty and bounded. Prove that a necessary and sufficient condition for $\Pi_K(\mathbf{x})$ to be a singleton, that is for $|\Pi_K(\mathbf{x})| = 1$, is for K to be convex.

6. Prove that for convex functions, $\nabla f(\mathbf{x}) \in \partial f(\mathbf{x})$, that is, the gradient belongs to the subgradient set.

7. Let $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ be a convex differentiable function and $\mathcal{K} \subseteq \mathbb{R}^n$ be a convex set. Prove that $\mathbf{x}^\star \in \mathcal{K}$ is a minimizer of f over $\mathcal{K}$ if and only if for any $\mathbf{y} \in \mathcal{K}$ it holds that $(\mathbf{y} - \mathbf{x}^\star)^\top \nabla f(\mathbf{x}^\star) \geq 0$.

8. Consider the n-dimensional simplex

$$\Delta_n = \{\mathbf{x} \in \mathbb{R}^n \mid \sum_{i=1}^{n} \mathbf{x}_i = 1,\, \mathbf{x}_i \geq 0 \ ,\ \forall i \in [n]\}.$$

Give an algorithm for computing the projection of a point $\mathbf{x} \in \mathbb{R}^n$ onto the set Δ_n (a near-linear time algorithm exists).

2.5 Bibliographic remarks

The reader is referred to dedicated books on convex optimization for much more in-depth treatment of the topics surveyed in this background chapter. For background in convex analysis see the texts [11, 68]. The classic textbook [12] gives a broad introduction to convex optimization with numerous applications. For an adaptive analysis of gradient descent with the Polyak stepsize see [33].

Chapter 3

Stochastic Gradient Descent

The most important optimization algorithm in the context of machine learning is stochastic gradient descent (SGD), especially for non-convex optimization and in the context of deep neural networks. In this chapter we spell out the algorithm and analyze it up to tight finite-time convergence rates.

3.1 Training feedforward neural networks

Perhaps the most common optimization problem in machine learning is that of training feedforward neural networks. In this problem, we are given a set of labelled data points, such as labelled images or text. Let $\{\mathbf{x}_i, y_i\}$ be the set of labelled data points, also called the training data.

The goal is to fit the weights of an artificial neural network in order to minimize the loss over the data. Mathematically, the feedforward network is a given weighted a-cyclic graph $G = (V, E, W)$. Each node v is assigned an activation function, which we assume is the same function for all nodes, denoted $\sigma : \mathbb{R}^d \mapsto \mathbb{R}$. Using a biological analogy, an activation function σ is a function that determines how strongly a neuron (i.e. a node) 'fires' for a given input by mapping the result into the desired range, usually $[0, 1]$ or $[-1, 1]$. Some popular examples include:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectified linear unit: $ReLU(x) = \max\{0, x\}$ (currently the most widely used of the three)

The inputs to the input layer nodes is a given data point, while the inputs to to all other nodes are the output of the nodes connected to it. We denote by $\rho(v)$ the set of input neighbors to node v. The top node output is the input to the loss function, which takes its "prediction" and the true label to form a loss.

For an input node v, its output as a function of the graph weights and input example $\mathbf{x}$ (of dimension d), which we denote as

$$v(W, \mathbf{x}) = \sigma\left(\sum_{i \in d} W_{v,i}\mathbf{x}_i\right)$$

The output of an internal node v is a function of its inputs $u \in \rho(v)$ and a given example $\mathbf{x}$, which we denote as

$$v(W, \mathbf{x}) = \sigma\left(\sum_{u \in \rho(v)} W_{uv}u(W, \mathbf{x})\right)$$

If we denote the top node as v^1, then the loss of the network over data point $(\mathbf{x}_i, y_i)$ is given by

$$\ell(v^1(W, \mathbf{x}_i), y_i).$$

The objective function becomes

$$f(W) = \mathop{\mathbf{E}}_{\mathbf{x}_i, y_i}\left[\ell(v^1(W, \mathbf{x}_i), y_i)\right]$$

For most commonly-used activation and loss functions, the above function is non-convex. However, it admits important computational properties. The most significant property is given in the following lemma.

Lemma 3.1 (Backpropagation lemma)**.** *The gradient of f can be computed in time $O(|E|)$.*

The proof of this lemma is left as an exercise, but we sketch the main ideas. For every variable W_{uv}, we have by linearity of expectation that

$$\frac{\partial}{\partial W_{uv}} f(W) = \mathop{\mathbf{E}}_{\mathbf{x}_i, y_i}\left[\frac{\partial}{\partial W_{uv}}\ell(v^1(W, \mathbf{x}_i), y_i)\right].$$

Next, using the chain rule, we claim that it suffices to know the partial derivatives of each node w.r.t. its immediate daughters. To see this, let us

write the derivative w.r.t. W_{uv} using the chain rule:

$$\begin{aligned}\frac{\partial}{\partial W_{uv}}\ell(v^1(W,\mathbf{x}_i),y_i) &= \frac{\partial \ell}{\partial v^1}\cdot\frac{\partial v^1}{\partial W_{uv}} \\ &= \frac{\partial \ell}{\partial v^1}\cdot\sum_{v^2\in\rho(v^1)}\frac{\partial v^1}{\partial v^2}\cdot\frac{\partial v_j}{\partial W_{uv}} = ... \\ &= \frac{\partial \ell}{\partial v^1}\cdot\sum_{v^2\in\rho(v^1)}\frac{\partial v^1}{\partial v^2}\cdot ... \cdot \sum_{v_j^k\in\rho(v^{k-1})}\cdot\frac{\partial v^k}{\partial W_{uv}}\end{aligned}$$

We conclude that we only need to obtain the E partial derivatives along the edges in order to compute all partial derivatives of the function. The actual product at each node can be computed by a dynamic program in linear time.

3.2 Gradient descent for smooth optimization

Before moving to stochastic gradient descent, we consider its deterministic counterpart: gradient descent, in the context of smooth non-convex optimization. Our notion of solution is a point with small gradient, i.e. $\|\nabla f(\mathbf{x})\| \le \varepsilon$.

As we prove below, this requires $O(\frac{1}{\varepsilon^2})$ iterations, each requiring one gradient computation. Recall that gradients can be computed efficiently, linear in the number of edges, in feed forward neural networks. Thus, the time to obtain a ε-approximate solution becomes $O(\frac{|E|m}{\varepsilon^2})$ for neural networks with E edges and over m examples.

Algorithm 2 Gradient descent

1: Input: f, T, initial point $\mathbf{x}_1 \in \mathcal{K}$, sequence of step sizes $\{\eta_t\}$
2: **for** $t = 1$ to T **do**
3: Let $\mathbf{y}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t),\ \mathbf{x}_{t+1} = \Pi_{\mathcal{K}}(\mathbf{y}_{t+1})$
4: **end for**
5: **return** $\mathbf{x}_{T+1}$

Although the choice of η_t can make a difference in practice, in theory the convergence of the vanilla GD algorithm is well understood and given in the following theorem. Below we assume that the function is bounded such that $|f(\mathbf{x})| \le M$.

Theorem 3.2. *For unconstrained minimization of β-smooth functions and $\eta_t = \frac{1}{\beta}$, GD Algorithm 2 converges as*

$$\frac{1}{T}\sum_t \|\nabla_t\|^2 \leq \frac{4M\beta}{T}.$$

Proof. Denote by ∇_t the shorthand for $\nabla f(\mathbf{x}_t)$, and $h_t = f(\mathbf{x}_t) - f(\mathbf{x}^*)$. The **Descent Lemma** is given in the following simple equation,

$$\begin{aligned} h_{t+1} - h_t &= f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) \\ &\leq \nabla_t^\top(\mathbf{x}_{t+1} - \mathbf{x}_t) + \frac{\beta}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 && \beta\text{-smoothness} \\ &= -\eta_t\|\nabla_t\|^2 + \frac{\beta}{2}\eta_t^2\|\nabla_t\|^2 && \text{algorithm defn.} \\ &= -\frac{1}{2\beta}\|\nabla_t\|^2 && \text{choice of } \eta_t = \frac{1}{\beta} \end{aligned}$$

Thus, summing up over T iterations, we have

$$\frac{1}{2\beta}\sum_{t=1}^{T}\|\nabla_t\|^2 \leq \sum_t (h_t - h_{t+1}) = h_1 - h_{T+1} \leq 2M$$

□

For convex functions, the above theorem implies convergence in function value due to the following lemma,

Lemma 3.3. *A convex function satisfies*

$$h_t \leq D\|\nabla_t\|,$$

and an α-strongly convex function satisfies

$$h_t \leq \frac{1}{2\alpha}\|\nabla_t\|^2.$$

Proof. The gradient upper bound for convex functions gives

$$h_t \leq \nabla_t(\mathbf{x}^* - \mathbf{x}_t) \leq D\|\nabla_t\|$$

The strongly convex case appears in Lemma 2.3.

□

3.3 Stochastic gradient descent

In the context of training feed forward neural networks, the key idea of Stochastic Gradient Descent is to modify the updates to be:

$$W_{t+1} = W_t - \eta \widetilde{\nabla}_t \tag{3.1}$$

where $\widetilde{\nabla}_t$ is a random variable with $\mathbf{E}[\widetilde{\nabla}_t] = \nabla f(W_t)$ and bounded second moment $\mathbf{E}[\|\widetilde{\nabla}_t\|_2^2] \leq \sigma^2$.

Luckily, getting the desired $\widetilde{\nabla}_t$ random variable is easy in the posed problem since the objective function is already in expectation form so:

$$\nabla f(W) = \nabla \mathop{\mathbf{E}}_{\mathbf{x}_i, y_i}[\ell(v^1(W, \mathbf{x}_i), y_i)] = \mathop{\mathbf{E}}_{\mathbf{x}_i, y_i}[\nabla \ell(v^1(W, \mathbf{x}_i), y_i)].$$

Therefore, at iteration t we can take $\widetilde{\nabla}_t = \nabla \ell(v^1(W, \mathbf{x}_i), y_i)$ where $i \in \{1, ..., m\}$ is picked uniformly at random. Based on the observation above, choosing $\widetilde{\nabla}_t$ this way preserves the desired expectation. So, for each iteration we only compute the gradient w.r.t. to one random example instead of the entire dataset, thereby drastically improving performance for every step. It remains to analyze how this impacts convergence.

Algorithm 3 Stochastic gradient descent

1: Input: f, T, initial point $\mathbf{x}_1 \in \mathcal{K}$, sequence of step sizes $\{\eta_t\}$
2: **for** $t = 1$ to T **do**
3: Let $\mathbf{y}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t),\ \mathbf{x}_{t+1} = \Pi_{\mathcal{K}}(\mathbf{y}_{t+1})$
4: **end for**
5: **return** $\mathbf{x}_{T+1}$

Theorem 3.4. *For unconstrained minimization of β-smooth functions and $\eta_t = \eta = \sqrt{\frac{M}{\beta \sigma^2 T}}$, SGD Algorithm 3 converges as*

$$\mathbf{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq 2\sqrt{\frac{M\beta\sigma^2}{T}}.$$

Proof. Denote by ∇_t the shorthand for $\nabla f(\mathbf{x}_t)$, and $h_t = f(\mathbf{x}_t) - f(\mathbf{x}^*)$. The stochastic descent lemma is given in the following equation,

$$\begin{aligned}
\mathbf{E}[h_{t+1} - h_t] &= \mathbf{E}[f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t)] \\
&\leq \mathbf{E}[\nabla_t^\top (\mathbf{x}_{t+1} - \mathbf{x}_t) + \frac{\beta}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2] && \beta\text{-smoothness} \\
&= -\mathbf{E}[\eta \nabla_t^\top \tilde{\nabla}_t] + \frac{\beta}{2}\eta^2 \mathbf{E}\,\|\tilde{\nabla}_t\|^2 && \text{algorithm defn.} \\
&= -\eta\|\nabla_t\|^2 + \frac{\beta}{2}\eta^2\sigma^2 && \text{variance bound.}
\end{aligned}$$

Thus, summing up over T iterations, we have for $\eta = \sqrt{\frac{M}{\beta\sigma^2 T}}$,

$$\begin{aligned}
\mathbf{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla_t\|^2\right] &\leq \tfrac{1}{T\eta}\textstyle\sum_t \mathbf{E}\left[h_t - h_{t+1}\right] + \eta\tfrac{\beta}{2}\sigma^2 \leq \tfrac{M}{T\eta} + \eta\tfrac{\beta}{2}\sigma^2 \\
&= \sqrt{\tfrac{M\beta\sigma^2}{T}} + \tfrac{1}{2}\sqrt{\tfrac{M\beta\sigma^2}{T}} \leq 2\sqrt{\tfrac{M\beta\sigma^2}{T}}.
\end{aligned}$$

□

We thus conclude that $O(\frac{1}{\varepsilon^4})$ iterations are needed to find a point with $\|\nabla f(\mathbf{x})\| \leq \varepsilon$, as opposed to $O(\frac{1}{\varepsilon^2})$. However, each iteration takes $O(|E|)$ time, instead of $O(|E|m)$ time for gradient descent.

This is why SGD is one of the most useful algorithms in machine learning.

3.4 Bibliographic remarks

For in depth treatment of backpropagation and the role of deep neural networks in machine learning the reader is referred to [25].

For detailed rigorous convergence proofs of first order methods, see lecture notes by Nesterov [57] and Nemirovskii [53, 54], as well as the recent text [13].

Chapter 4

Generalization and Non-Smooth Optimization

In previous chapter we have introduced the framework of mathematical optimization within the context of machine learning. We have described the mathematical formulation of several machine learning problems, notably training neural networks, as optimization problems. We then described as well as analyzed the most useful optimization method to solve such formulations: stochastic gradient descent.

However, several important questions arise:

1. SGD was analyzed for smooth functions. Can we minimize non-smooth objectives?

2. Given an ERM problem (a.k.a. learning from examples, see first chapter), what can we say about generalization to unseen examples? How does it affect optimization?

3. Are there faster algorithms than SGD in the context of ML?

In this chapter we address the first two, and devote the rest of this manuscript/course to the last question.

How many examples are needed to learn a certain concept? This is a fundamental question of statistical/computational learning theory that has been studied for decades (see end of chapter for bibliographic references).

The classical setting of learning from examples is statistical. It assumes examples are drawn i.i.d from a fixed, arbitrary and unknown distribution. The mathematical optimization formulations that we have derived for the ERM problem assume that we have sufficiently many examples, such that

optimizing a certain predictor/neural-network/machine on them will result in a solution that is capable of generalizing to unseen examples. The number of examples needed to generalize is called the *sample complexity* of the problem, and it depends on the concept we are learning as well as the hypothesis class over which we are trying to optimize.

There are dimensionality notions in the literature, notably the VC-dimension and related notions, that give precise bounds on the sample complexity for various hypothesis classes. In this text we take an algorithmic approach, which is also deterministic. Instead of studying sample complexity, which is non-algorithmic, we study algorithms for regret minimization. We will show that they imply generalization for a broad class of machines.

4.1 A note on non-smooth optimization

Minimization of a function that is both non-convex and non-smooth is in general hopeless, from an information theoretic perspective. The following image explains why. The depicted function on the interval $[0, 1]$ has a single local/global minimum, and if the crevasse is narrow enough, it cannot be found by any method other than extensive brute-force search, which can take arbitrarily long.

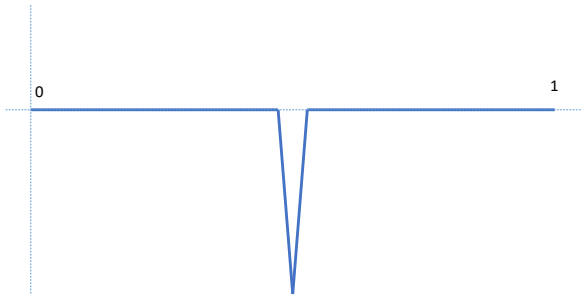

Figure 4.1: Intractability of nonsmooth optimization

Since non-convex and non-smooth optimization is hopeless, in the context of non-smooth functions we only consider **convex** optimization.

4.2 Minimizing Regret

The setting we consider for the rest of this chapter is that of online (convex) optimization. In this setting a learner iteratively predicts a point $\mathbf{x}_t \in \mathcal{K}$ in a convex set $\mathcal{K} \subseteq \mathbb{R}^d$, and then receives a cost according to an adversarially chosen convex function $f_t \in \mathcal{F}$ from family $\mathcal{F}$.

The goal of the algorithms introduced in this chapter is to minimize worst-case *regret*, or difference between total cost and that of best point in hindsight:

$$\text{regret} = \sup_{f_1,\ldots,f_T \in \mathcal{F}} \left\{ \sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\mathbf{x}) \right\}.$$

In order to compare regret to optimization error it is useful to consider the average regret, or regret/T. Let $\bar{\mathbf{x}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t$ be the average decision. If the functions f_t are all equal to a single function $f : \mathcal{K} \mapsto \mathbb{R}$, then Jensen's inequality implies that $f(\bar{\mathbf{x}}_T)$ converges to $f(\mathbf{x}^\star)$ if the average regret is vanishing, since

$$f(\bar{\mathbf{x}}_T) - f(\mathbf{x}^\star) \leq \frac{1}{T}\sum_{t=1}^{T}[f(\mathbf{x}_t) - f(\mathbf{x}^\star)] = \frac{\text{regret}}{T}$$

4.3 Regret implies generalization

Statistical learning theory for learning from examples postulates that examples from a certain concept are sampled i.i.d. from a fixed and unknown distribution. The learners' goal is to choose a hypothesis from a certain hypothesis class that can generalize to unseen examples.

More formally, let $\mathcal{D}$ be a distribution over labelled examples $\{\mathbf{a}_i \in \mathbb{R}^d, b_i \in \mathbb{R}\} \sim \mathcal{D}$. Let $\mathcal{H} = \{\mathbf{x}\}$, $\mathbf{x} : \mathbb{R}^d \mapsto \mathbb{R}$ be a hypothsis class over which we are trying to learn (such as linear separators, deep neural networks, etc.). The *generalization error* of a hypothesis is the expected error of a hypothesis over randomly chosen examples according to a given loss function $\ell : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, which is applied to the prediction of the hypothesis and the true label, $\ell(\mathbf{x}(\mathbf{a}_i), b_i)$. Thus,

$$\text{error}(\mathbf{x}) = \mathop{\mathbf{E}}_{\mathbf{a}_i, b_i \sim \mathcal{D}}[\ell(\mathbf{x}(\mathbf{a}_i), b_i)].$$

An algorithm that attains sublinear regret over the hypothesis class $\mathcal{H}$, w.r.t. loss functions given by $f_t(\mathbf{x}) = f_{\mathbf{a},b}(\mathbf{x}) = \ell(\mathbf{x}(\mathbf{a}), b)$, gives rise to a generalizing hypothesis as follows.

Lemma 4.1. *Let $\bar{\mathbf{x}} = \mathbf{x}_t$ for $t \in [T]$ be chose uniformly at random from $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$. Then, with expectation taken over random choice of $\bar{\mathbf{x}}$ as well as choices of $f_t \sim \mathcal{D}$,*

$$\mathbf{E}[\text{error}(\bar{\mathbf{x}})] \leq \mathbf{E}[\text{error}(\mathbf{x}^*)] + \frac{\text{regret}}{T}$$

Proof. By random choice of $\bar{\mathbf{x}}$, we have

$$\mathbf{E}[f(\bar{\mathbf{x}})] = \mathbf{E}\left[\frac{1}{T}\sum_t f(\mathbf{x}_t)\right]$$

Using the fact that $f_t \sim \mathcal{D}$, we have

$$\begin{aligned}
\mathbf{E}[\text{error}(\bar{\mathbf{x}})] &= \mathbf{E}_{f\sim\mathcal{D}}[f(\bar{\mathbf{x}})] \\
&= \mathbf{E}_{f_t}[\tfrac{1}{T}\textstyle\sum_t f_t(\mathbf{x}_t)] \\
&\leq \mathbf{E}_{f_t}[\tfrac{1}{T}\textstyle\sum_t f_t(\mathbf{x}^\star)] + \tfrac{\text{regret}}{T} \\
&= \mathbf{E}_f[f(\mathbf{x}^\star)] + \tfrac{\text{regret}}{T} \\
&= \mathbf{E}_f[\text{error}(\mathbf{x}^\star)] + \tfrac{\text{regret}}{T}
\end{aligned}$$

□

4.4 Online gradient descent

Perhaps the simplest algorithm that applies to the most general setting of online convex optimization is online gradient descent. This algorithm is an online version of standard gradient descent for offline optimization we have seen in the previous chapter. Pseudo-code for the algorithm is given in Algorithm 4, and a conceptual illustration is given in Figure 4.2.

In each iteration, the algorithm takes a step from the previous point in the direction of the gradient of the previous cost. This step may result in a point outside of the underlying convex set. In such cases, the algorithm projects the point back to the convex set, i.e. finds its closest point in the convex set. Despite the fact that the next cost function may be completely different than the costs observed thus far, the regret attained by the algorithm is sublinear. This is formalized in the following theorem (recall the definition of G and D from the previous chapter).

Theorem 4.2. *Online gradient descent with step sizes $\{\eta_t = \frac{D}{G\sqrt{t}},\ t \in [T]\}$ guarantees the following for all $T \geq 1$:*

$$\text{regret}_T = \sum_{t=1}^T f_t(\mathbf{x}_t) - \min_{\mathbf{x}^\star \in \mathcal{K}} \sum_{t=1}^T f_t(\mathbf{x}^\star) \ \leq 3GD\sqrt{T}$$

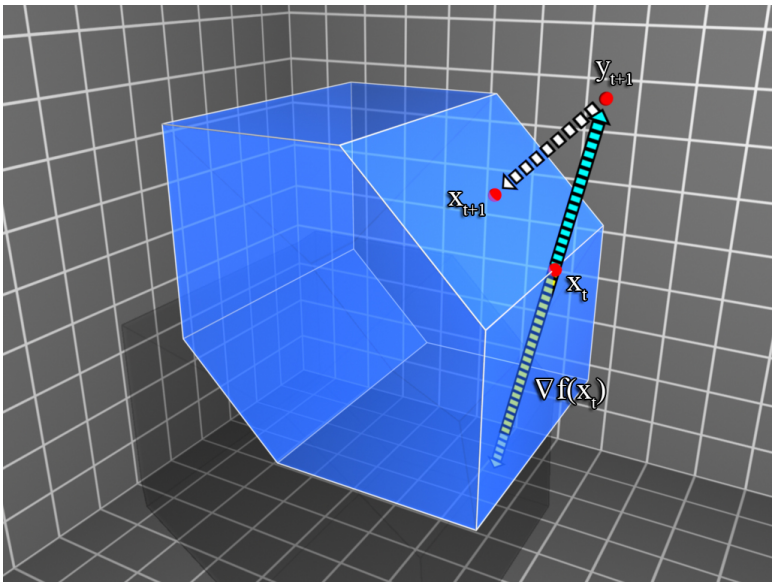

Figure 4.2: Online gradient descent: the iterate $\mathbf{x}_{t+1}$ is derived by advancing $\mathbf{x}_t$ in the direction of the current gradient ∇_t, and projecting back into $\mathcal{K}$.

Algorithm 4 online gradient descent

1: Input: convex set $\mathcal{K}$, T, $\mathbf{x}_1 \in \mathcal{K}$, step sizes $\{\eta_t\}$
2: **for** $t = 1$ to T **do**
3: Play $\mathbf{x}_t$ and observe cost $f_t(\mathbf{x}_t)$.
4: Update and project:

$$\mathbf{y}_{t+1} = \mathbf{x}_t - \eta_t \nabla f_t(\mathbf{x}_t)$$
$$\mathbf{x}_{t+1} = \prod_{\mathcal{K}}(\mathbf{y}_{t+1})$$

5: **end for**

Proof. Let $\mathbf{x}^\star \in \arg\min_{\mathbf{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\mathbf{x})$. Define $\nabla_t \triangleq \nabla f_t(\mathbf{x}_t)$. By convexity

$$f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star) \le \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star) \tag{4.1}$$

We first upper-bound $\nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star)$ using the update rule for $\mathbf{x}_{t+1}$ and Theorem 2.1 (the Pythagorean theorem):

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 = \left\| \prod_{\mathcal{K}}(\mathbf{x}_t - \eta_t \nabla_t) - \mathbf{x}^\star \right\|^2 \le \|\mathbf{x}_t - \eta_t \nabla_t - \mathbf{x}^\star\|^2 \tag{4.2}$$

Hence,

$$\begin{aligned} \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 &\leq \|\mathbf{x}_t - \mathbf{x}^\star\|^2 + \eta_t^2 \|\nabla_t\|^2 - 2\eta_t \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star) \\ 2\nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star) &\leq \frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} + \eta_t G^2 \end{aligned} \tag{4.3}$$

Summing (4.1) and (4.3) from $t = 1$ to T, and setting $\eta_t = \frac{D}{G\sqrt{t}}$ (with $\frac{1}{\eta_0} \triangleq 0$):

$$\begin{aligned} &2\left(\sum_{t=1}^T f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star)\right) \leq 2\sum_{t=1}^T \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star) \\ &\leq \sum_{t=1}^T \frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} + G^2 \sum_{t=1}^T \eta_t \\ &\leq \sum_{t=1}^T \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) + G^2 \sum_{t=1}^T \eta_t && \frac{1}{\eta_0} \triangleq 0, \\ & && \|\mathbf{x}_{T+1} - \mathbf{x}^*\|^2 \geq 0 \\ &\leq D^2 \sum_{t=1}^T \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) + G^2 \sum_{t=1}^T \eta_t \\ &\leq D^2 \frac{1}{\eta_T} + G^2 \sum_{t=1}^T \eta_t && \text{telescoping series} \\ &\leq 3DG\sqrt{T}. \end{aligned}$$

The last inequality follows since $\eta_t = \frac{D}{G\sqrt{t}}$ and $\sum_{t=1}^T \frac{1}{\sqrt{t}} \leq 2\sqrt{T}$. □

The online gradient descent algorithm is straightforward to implement, and updates take linear time given the gradient. However, there is a projection step which may take significantly longer.

4.5 Lower bounds

Theorem 4.3. *Any algorithm for online convex optimization incurs $\Omega(DG\sqrt{T})$ regret in the worst case. This is true even if the cost functions are generated from a fixed stationary distribution.*

We give a sketch of the proof; filling in all details is left as an exercise at the end of this chapter.

Consider an instance of OCO where the convex set $\mathcal{K}$ is the n-dimensional hypercube, i.e.

$$\mathcal{K} = \{\mathbf{x} \in \mathbb{R}^n \ , \ \|\mathbf{x}\|_\infty \leq 1\}.$$

There are 2^n linear cost functions, one for each vertex $\mathbf{v} \in \{\pm 1\}^n$, defined as

$$\forall \mathbf{v} \in \{\pm 1\}^n \ , \ f_{\mathbf{v}}(\mathbf{x}) = \mathbf{v}^\top \mathbf{x}.$$

Notice that both the diameter of $\mathcal{K}$ and the bound on the norm of the cost function gradients, denoted G, are bounded by

$$D \leq \sqrt{\sum_{i=1}^{n} 2^2} = 2\sqrt{n}, \ G = \sqrt{\sum_{i=1}^{n} (\pm 1)^2} = \sqrt{n}$$

The cost functions in each iteration are chosen at random, with uniform probability, from the set $\{f_{\mathbf{v}}, \mathbf{v} \in \{\pm 1\}^n\}$. Denote by $\mathbf{v}_t \in \{\pm 1\}^n$ the vertex chosen in iteration t, and denote $f_t = f_{\mathbf{v}_t}$. By uniformity and independence, for any t and $\mathbf{x}_t$ chosen online, $\mathbf{E}_{\mathbf{v}_t}[f_t(\mathbf{x}_t)] = \mathbf{E}_{\mathbf{v}_t}[\mathbf{v}_t^\top \mathbf{x}_t] = 0$. However,

$$\begin{aligned}
\mathop{\mathbf{E}}_{\mathbf{v}_1,\ldots,\mathbf{v}_T} \left[\min_{\mathbf{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\mathbf{x}) \right] &= \mathbf{E} \left[\min_{\mathbf{x}\in\mathcal{K}} \sum_{i\in[n]} \sum_{t=1}^{T} \mathbf{v}_t(i) \cdot \mathbf{x}_i \right] \\
&= n\, \mathbf{E} \left[- \left| \sum_{t=1}^{T} \mathbf{v}_t(1) \right| \right] \qquad \text{i.i.d. coordinates} \\
&= -\Omega(n\sqrt{T}).
\end{aligned}$$

The last equality is left as exercise 3.

The facts above nearly complete the proof of Theorem 4.3; see the exercises at the end of this chapter.

4.6 Online gradient descent for strongly convex functions

The first algorithm that achieves regret logarithmic in the number of iterations is a twist on the online gradient descent algorithm, changing only the step size. The following theorem establishes logarithmic bounds on the regret if the cost functions are strongly convex.

Theorem 4.4. *For α-strongly convex loss functions, online gradient descent with step sizes $\eta_t = \frac{1}{\alpha t}$ achieves the following guarantee for all $T \geq 1$*

$$\text{regret}_T \;\leq\; \sum_{t=1}^{T} \frac{1}{\alpha t}\|\nabla_t\|^2 \leq \frac{G^2}{2\alpha}(1+\log T).$$

Proof. Let $\mathbf{x}^\star \in \arg\min_{\mathbf{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\mathbf{x})$. Recall the definition of regret

$$\text{regret}_T \;= \sum_{t=1}^{T} f_t(\mathbf{x}_t) - \sum_{t=1}^{T} f_t(\mathbf{x}^\star).$$

Define $\nabla_t \triangleq \nabla f_t(\mathbf{x}_t)$. Applying the definition of α-strong convexity to the pair of points $\mathbf{x}_t$,$\mathbf{x}^*$, we have

$$2(f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star)) \;\leq\; 2\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star) - \alpha\|\mathbf{x}^\star - \mathbf{x}_t\|^2. \tag{4.4}$$

We proceed to upper-bound $\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star)$. Using the update rule for $\mathbf{x}_{t+1}$ and the Pythagorean theorem 2.1, we get

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \;=\; \|\prod_{\mathcal{K}}(\mathbf{x}_t - \eta_t\nabla_t) - \mathbf{x}^\star\|^2 \leq \|\mathbf{x}_t - \eta_t\nabla_t - \mathbf{x}^\star\|^2.$$

Hence,

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \quad\leq\quad \|\mathbf{x}_t - \mathbf{x}^\star\|^2 + \eta_t^2\|\nabla_t\|^2 - 2\eta_t\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star)$$

and

$$2\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star) \quad\leq\quad \frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} + \eta_t\|\nabla_t\|^2. \tag{4.5}$$

Summing (4.5) from $t = 1$ to T, setting $\eta_t = \frac{1}{\alpha t}$ (define $\frac{1}{\eta_0} \triangleq 0$), and combining with (4.4), we have:

$$\begin{aligned} & 2\sum_{t=1}^{T}(f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star)) \\ \leq \quad & \sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} - \alpha\right) + \sum_{t=1}^{T} \eta_t \|\nabla_t\|^2 \\ & \text{since } \frac{1}{\eta_0} \triangleq 0, \|\mathbf{x}_{T+1} - \mathbf{x}^*\|^2 \geq 0 \\ = \quad & 0 + \sum_{t=1}^{T} \frac{1}{\alpha t} \|\nabla_t\|^2 \\ \leq \quad & \frac{G^2}{\alpha}(1 + \log T) \end{aligned}$$

□

4.7 Online Gradient Descent implies SGD

In this section we notice that OGD and its regret bounds imply the SGD bounds we have studied in the previous chapter. The main advantage are the guarantees for non-smooth stochastic optimization, and constrained optimization.

Recall that in stochastic optimization, the optimizer attempts to minimize a convex function over a convex domain as given by the mathematical program:

$$\min_{\mathbf{x} \in \mathcal{K}} f(\mathbf{x}).$$

However, unlike standard offline optimization, the optimizer is given access to a noisy gradient oracle, defined by

$$\mathcal{O}(\mathbf{x}) \triangleq \tilde{\nabla}_{\mathbf{x}} \quad \text{s.t.} \quad \mathbf{E}[\tilde{\nabla}_{\mathbf{x}}] = \nabla f(\mathbf{x}) \ , \ \mathbf{E}[\|\tilde{\nabla}_{\mathbf{x}}\|^2] \leq G^2$$

That is, given a point in the decision set, a noisy gradient oracle returns a random vector whose expectation is the gradient at the point and whose second moment is bounded by G^2.

We will show that regret bounds for OCO translate to convergence rates for stochastic optimization. As a special case, consider the online gradient

descent algorithm whose regret is bounded by

$$\text{regret}_T = O(DG\sqrt{T})$$

Applying the OGD algorithm over a sequence of linear functions that are defined by the noisy gradient oracle at consecutive points, and finally returning the average of all points along the way, we obtain the stochastic gradient descent algorithm, presented in Algorithm 5.

Algorithm 5 stochastic gradient descent

1: Input: f, $\mathcal{K}$, T, $\mathbf{x}_1 \in \mathcal{K}$, step sizes $\{\eta_t\}$
2: **for** $t = 1$ to T **do**
3: Let $\tilde{\nabla}_t = \mathcal{O}(\mathbf{x}_t)$ and define: $f_t(\mathbf{x}) \triangleq \langle \tilde{\nabla}_t, \mathbf{x} \rangle$
4: Update and project:

$$\mathbf{y}_{t+1} = \mathbf{x}_t - \eta_t \tilde{\nabla}_t$$

$$\mathbf{x}_{t+1} = \prod_{\mathcal{K}} (\mathbf{y}_{t+1})$$

5: **end for**
6: **return** $\bar{\mathbf{x}}_T \triangleq \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}_t$

Theorem 4.5. *Algorithm 5 with step sizes $\eta_t = \frac{D}{G\sqrt{t}}$ guarantees*

$$\mathbf{E}[f(\bar{\mathbf{x}}_T)] \leq \min_{\mathbf{x}^\star \in \mathcal{K}} f(\mathbf{x}^\star) + \frac{3GD}{\sqrt{T}}$$

Proof. By the regret guarantee of OGD, we have

$$\begin{aligned}
&\mathbf{E}[f(\bar{\mathbf{x}}_T)] - f(\mathbf{x}^\star) \\
&\leq \mathbf{E}[\frac{1}{T} \sum_t f(\mathbf{x}_t)] - f(\mathbf{x}^\star) && \text{convexity of } f \text{ (Jensen)} \\
&\leq \frac{1}{T} \mathbf{E}[\sum_t \langle \nabla f(\mathbf{x}_t), \mathbf{x}_t - \mathbf{x}^\star \rangle] && \text{convexity again} \\
&= \frac{1}{T} \mathbf{E}[\sum_t \langle \tilde{\nabla}_t, \mathbf{x}_t - \mathbf{x}^\star \rangle] && \text{noisy gradient estimator} \\
&= \frac{1}{T} \mathbf{E}[\sum_t f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star)] && \text{Algorithm 5, line (3)} \\
&\leq \frac{\text{regret}_T}{T} && \text{definition} \\
&\leq \frac{3GD}{\sqrt{T}} && \text{theorem 4.2}
\end{aligned}$$

□

It is important to note that in the proof above, we have used the fact that the regret bounds of online gradient descent hold against an adaptive adversary. This need arises since the cost functions f_t defined in Algorithm 5 depend on the choice of decision $\mathbf{x}_t \in \mathcal{K}$.

In addition, the careful reader may notice that by plugging in different step sizes (also called learning rates) and applying SGD to strongly convex functions, one can attain $\tilde{O}(1/T)$ convergence rates. Details of this derivation are left as exercise 1.

4.8 Exercises

1. Prove that SGD for a strongly convex function can, with appropriate parameters η_t, converge as $\tilde{O}(\frac{1}{T})$. You may assume that the gradient estimators have Euclidean norms bounded by the constant G.

2. Design an OCO algorithm that attains the same asymptotic regret bound as OGD, up to factors logarithmic in G and D, without knowing the parameters G and D ahead of time.

3. In this exercise we prove a tight lower bound on the regret of any algorithm for online convex optimization.

   (a) For any sequence of T fair coin tosses, let N_h be the number of head outcomes and N_t be the number of tails. Give an asymptotically tight upper and lower bound on $\mathbf{E}[|N_h - N_t|]$ (i.e., order of growth of this random variable as a function of T, up to multiplicative and additive constants).

   (b) Consider a 2-expert problem, in which the losses are inversely correlated: either expert one incurs a loss of one and the second expert zero, or vice versa. Use the fact above to design a setting in which any experts algorithm incurs regret asymptotically matching the upper bound.

   (c) Consider the general OCO setting over a convex set $\mathcal{K}$. Design a setting in which the cost functions have gradients whose norm is bounded by G, and obtain a lower bound on the regret as a function of G, the diameter of $\mathcal{K}$, and the number of game iterations.

4.9 Bibliographic remarks

The OCO framework was introduced by Zinkevich in [87], where the OGD algorithm was introduced and analyzed. Precursors to this algorithm, albeit for less general settings, were introduced and analyzed in [47]. Logarithmic regret algorithms for Online Convex Optimization were introduced and analyzed in [32]. For more detailed exposition on this prediction framework and its applications see [31].

The SGD algorithm dates back to Robbins and Monro [67]. Application of SGD to soft-margin SVM training was explored in [74]. Tight convergence rates of SGD for strongly convex and non-smooth functions were only recently obtained in [35],[62],[76].

Chapter 5

Regularization

In this chapter we consider a generalization of the gradient descent called by different names in different communities (such as mirrored-descent, or regularized-follow-the-leader). The common theme of this generalization is called *Regularization*, a concept that is founded in generalization theory. Since this course focuses on optimization rather than generalization, we shall refer the reader to the generalization aspect of regularization, and focus hereby on optimization algorithms.

We start by motivating this general family of methods using the fundamental problem of decision theory.

5.1 Motivation: prediction from expert advice

Consider the following fundamental iterative decision making problem:

At each time step $t = 1, 2, \ldots, T$, the decision maker faces a choice between two actions A or B (i.e., buy or sell a certain stock). The decision maker has assistance in the form of N "experts" that offer their advice. After a choice between the two actions has been made, the decision maker receives feedback in the form of a loss associated with each decision. For simplicity one of the actions receives a loss of zero (i.e., the "correct" decision) and the other a loss of one.

We make the following elementary observations:

1. A decision maker that chooses an action uniformly at random each iteration, trivially attains a loss of $\frac{T}{2}$ and is "correct" 50% of the time.

2. In terms of the number of mistakes, no algorithm can do better in the worst case! In a later exercise, we will devise a randomized setting in which the expected number of mistakes of any algorithm is at least $\frac{T}{2}$.

We are thus motivated to consider a *relative performance metric*: can the decision maker make as few mistakes as the best expert in hindsight? The next theorem shows that the answer in the worst case is negative for a deterministic decision maker.

Theorem 5.1. *Let $L \leq \frac{T}{2}$ denote the number of mistakes made by the best expert in hindsight. Then there does not exist a deterministic algorithm that can guarantee less than $2L$ mistakes.*

Proof. Assume that there are only two experts and one always chooses option A while the other always chooses option B. Consider the setting in which an adversary always chooses the opposite of our prediction (she can do so, since our algorithm is deterministic). Then, the total number of mistakes the algorithm makes is T. However, the best expert makes no more than $\frac{T}{2}$ mistakes (at every iteration exactly one of the two experts is mistaken). Therefore, there is no algorithm that can always guarantee less than $2L$ mistakes.

□

This observation motivates the design of random decision making algorithms, and indeed, the OCO framework gracefully models decisions on a continuous probability space. Henceforth we prove Lemmas 5.3 and 5.4 that show the following:

Theorem 5.2. *Let $\varepsilon \in (0, \frac{1}{2})$. Suppose the best expert makes L mistakes. Then:*

1. *There is an efficient deterministic algorithm that can guarantee less than $2(1+\varepsilon)L + \frac{2\log N}{\varepsilon}$ mistakes;*

2. *There is an efficient randomized algorithm for which the expected number of mistakes is at most $(1+\varepsilon)L + \frac{\log N}{\varepsilon}$.*

5.1.1 The weighted majority algorithm

The weighted majority (WM) algorithm is intuitive to describe: each expert i is assigned a weight $W_t(i)$ at every iteration t. Initially, we set $W_1(i) = 1$ for all experts $i \in [N]$. For all $t \in [T]$ let $S_t(A), S_t(B) \subseteq [N]$ be the set of experts that choose A (and respectively B) at time t. Define,

$$W_t(A) = \sum_{i \in S_t(A)} W_t(i) \qquad W_t(B) = \sum_{i \in S_t(B)} W_t(i)$$

and predict according to

$$a_t = \begin{cases} A & \text{if } W_t(A) \geq W_t(B) \\ B & \text{otherwise.} \end{cases}$$

Next, update the weights $W_t(i)$ as follows:

$$W_{t+1}(i) = \begin{cases} W_t(i) & \text{if expert } i \text{ was correct} \\ W_t(i)(1-\varepsilon) & \text{if expert } i \text{ was wrong} \end{cases},$$

where ε is a parameter of the algorithm that will affect its performance. This concludes the description of the WM algorithm. We proceed to bound the number of mistakes it makes.

Lemma 5.3. *Denote by M_t the number of mistakes the algorithm makes until time t, and by $M_t(i)$ the number of mistakes made by expert i until time t. Then, for any expert $i \in [N]$ we have*

$$M_T \leq 2(1+\varepsilon)M_T(i) + \frac{2\log N}{\varepsilon}.$$

We can optimize ε to minimize the above bound. The expression on the right hand side is of the form $f(x) = ax + b/x$, that reaches its minimum at $x = \sqrt{b/a}$. Therefore the bound is minimized at $\varepsilon^\star = \sqrt{\log N / M_T(i)}$. Using this optimal value of ε, we get that for the best expert $i^\star$

$$M_T \leq 2M_T(i^\star) + O\left(\sqrt{M_T(i^\star)\log N}\right).$$

Of course, this value of $\varepsilon^\star$ cannot be used in advance since we do not know which expert is the best one ahead of time (and therefore we do not know the value of $M_T(i^\star)$). However, we shall see later on that the same asymptotic bound can be obtained even without this prior knowledge.

Let us now prove Lemma 5.3.

Proof. Let $\Phi_t = \sum_{i=1}^{N} W_t(i)$ for all $t \in [T]$, and note that $\Phi_1 = N$.

Notice that $\Phi_{t+1} \leq \Phi_t$. However, on iterations in which the WM algorithm erred, we have

$$\Phi_{t+1} \leq \Phi_t(1 - \frac{\varepsilon}{2}),$$

the reason being that experts with at least half of total weight were wrong (else WM would not have erred), and therefore

$$\Phi_{t+1} \leq \frac{1}{2}\Phi_t(1-\varepsilon) + \frac{1}{2}\Phi_t = \Phi_t(1 - \frac{\varepsilon}{2}).$$

From both observations,

$$\Phi_t \leq \Phi_1(1 - \frac{\varepsilon}{2})^{M_t} = N(1 - \frac{\varepsilon}{2})^{M_t}.$$

On the other hand, by definition we have for any expert i that

$$W_T(i) = (1-\varepsilon)^{M_T(i)}.$$

Since the value of $W_T(i)$ is always less than the sum of all weights Φ_T, we conclude that

$$(1-\varepsilon)^{M_T(i)} = W_T(i) \leq \Phi_T \leq N(1 - \frac{\varepsilon}{2})^{M_T}.$$

Taking the logarithm of both sides we get

$$M_T(i)\log(1-\varepsilon) \leq \log N + M_T \log(1 - \frac{\varepsilon}{2}).$$

Next, we use the approximations

$$-x - x^2 \leq \log(1-x) \leq -x \qquad 0 < x < \frac{1}{2},$$

which follow from the Taylor series of the logarithm function, to obtain that

$$-M_T(i)(\varepsilon + \varepsilon^2) \leq \log N - M_T\frac{\varepsilon}{2},$$

and the lemma follows. □

5.1.2 Randomized weighted majority

In the randomized version of the WM algorithm, denoted RWM, we choose expert i w.p. $p_t(i) = W_t(i)/\sum_{j=1}^{N} W_t(j)$ at time t.

Lemma 5.4. *Let M_t denote the number of mistakes made by RWM until iteration t. Then, for any expert $i \in [N]$ we have*

$$\mathbf{E}[M_T] \leq (1+\varepsilon)M_T(i) + \frac{\log N}{\varepsilon}.$$

The proof of this lemma is very similar to the previous one, where the factor of two is saved by the use of randomness:

Proof. As before, let $\Phi_t = \sum_{i=1}^{N} W_t(i)$ for all $t \in [T]$, and note that $\Phi_1 = N$. Let $\tilde{m}_t = M_t - M_{t-1}$ be the indicator variable that equals one if the RWM algorithm makes a mistake on iteration t. Let $m_t(i)$ equal one if the i'th expert makes a mistake on iteration t and zero otherwise. Inspecting the sum of the weights:

$$\begin{aligned}
\Phi_{t+1} &= \sum_i W_t(i)(1-\varepsilon m_t(i)) \\
&= \Phi_t(1-\varepsilon \sum_i p_t(i)m_t(i)) && p_t(i) = \tfrac{W_t(i)}{\sum_j W_t(j)} \\
&= \Phi_t(1-\varepsilon\,\mathbf{E}[\tilde{m}_t]) \\
&\leq \Phi_t e^{-\varepsilon\,\mathbf{E}[\tilde{m}_t]}. && 1+x \leq e^x
\end{aligned}$$

On the other hand, by definition we have for any expert i that

$$W_T(i) = (1-\varepsilon)^{M_T(i)}$$

Since the value of $W_T(i)$ is always less than the sum of all weights Φ_T, we conclude that

$$(1-\varepsilon)^{M_T(i)} = W_T(i) \leq \Phi_T \leq N e^{-\varepsilon\,\mathbf{E}[M_T]}.$$

Taking the logarithm of both sides we get

$$M_T(i)\log(1-\varepsilon) \leq \log N - \varepsilon\,\mathbf{E}[M_T]$$

Next, we use the approximation

$$-x - x^2 \leq \log(1-x) \leq -x \qquad , \quad 0 < x < \frac{1}{2}$$

to obtain

$$-M_T(i)(\varepsilon + \varepsilon^2) \leq \log N - \varepsilon \, \mathbf{E}[M_T],$$

and the lemma follows. □

5.1.3 Hedge

The RWM algorithm is in fact more general: instead of considering a discrete number of mistakes, we can consider measuring the performance of an expert by a non-negative real number $\ell_t(i)$, which we refer to as the *loss* of the expert i at iteration t. The randomized weighted majority algorithm guarantees that a decision maker following its advice will incur an average expected loss approaching that of the best expert in hindsight.

Historically, this was observed by a different and closely related algorithm called Hedge.

Algorithm 6 Hedge

1: Initialize: $\forall i \in [N],\ W_1(i) = 1$
2: **for** $t = 1$ to T **do**
3: Pick $i_t \sim_R W_t$, i.e., $i_t = i$ with probability $\mathbf{x}_t(i) = \frac{W_t(i)}{\sum_j W_t(j)}$
4: Incur loss $\ell_t(i_t)$.
5: Update weights $W_{t+1}(i) = W_t(i)e^{-\varepsilon \ell_t(i)}$
6: **end for**

Henceforth, denote in vector notation the expected loss of the algorithm by

$$\mathbf{E}[\ell_t(i_t)] = \sum_{i=1}^{N} \mathbf{x}_t(i)\ell_t(i) = \mathbf{x}_t^\top \ell_t$$

Theorem 5.5. *Let ℓ_t^2 denote the N-dimensional vector of square losses, i.e., $\ell_t^2(i) = \ell_t(i)^2$, let $\varepsilon > 0$, and assume all losses to be non-negative. The Hedge algorithm satisfies for any expert $i^\star \in [N]$:*

$$\sum_{t=1}^{T} \mathbf{x}_t^\top \ell_t \leq \sum_{t=1}^{T} \ell_t(i^\star) + \varepsilon \sum_{t=1}^{T} \mathbf{x}_t^\top \ell_t^2 + \frac{\log N}{\varepsilon}$$

Proof. As before, let $\Phi_t = \sum_{i=1}^{N} W_t(i)$ for all $t \in [T]$, and note that $\Phi_1 = N$.

Inspecting the sum of weights:

$$\begin{aligned}
\Phi_{t+1} &= \textstyle\sum_i W_t(i) e^{-\varepsilon \ell_t(i)} \\
&= \Phi_t \textstyle\sum_i \mathbf{x}_t(i) e^{-\varepsilon \ell_t(i)} && \mathbf{x}_t(i) = \frac{W_t(i)}{\sum_j W_t(j)} \\
&\leq \Phi_t \textstyle\sum_i \mathbf{x}_t(i)(1 - \varepsilon \ell_t(i) + \varepsilon^2 \ell_t(i)^2)) && \text{for } x \geq 0, \\
& && e^{-x} \leq 1 - x + x^2 \\
&= \Phi_t (1 - \varepsilon \mathbf{x}_t^\top \ell_t + \varepsilon^2 \mathbf{x}_t^\top \ell_t^2) \\
&\leq \Phi_t e^{-\varepsilon \mathbf{x}_t^\top \ell_t + \varepsilon^2 \mathbf{x}_t^\top \ell_t^2}. && 1 + x \leq e^x
\end{aligned}$$

On the other hand, by definition, for expert $i^\star$ we have that

$$W_T(i^\star) = e^{-\varepsilon \sum_{t=1}^{T} \ell_t(i^\star)}$$

Since the value of $W_T(i^\star)$ is always less than the sum of all weights Φ_t, we conclude that

$$W_T(i^\star) \leq \Phi_T \leq N e^{-\varepsilon \sum_t \mathbf{x}_t^\top \ell_t + \varepsilon^2 \sum_t \mathbf{x}_t^\top \ell_t^2}.$$

Taking the logarithm of both sides we get

$$-\varepsilon \sum_{t=1}^{T} \ell_t(i^\star) \leq \log N - \varepsilon \sum_{t=1}^{T} \mathbf{x}_t^\top \ell_t + \varepsilon^2 \sum_{t=1}^{T} \mathbf{x}_t^\top \ell_t^2$$

and the theorem follows by simplifying. □

5.2 The Regularization framework

In the previous section we studied the multiplicative weights update method for decision making. A natural question is: couldn't we have used online gradient descent for the same exact purpose?

Indeed, the setting of prediction from expert advice naturally follows into the framework of online convex optimization. To see this, consider the loss functions given by

$$f_t(\mathbf{x}) = \ell_t^\top \mathbf{x} = \mathop{\mathbf{E}}_{i \sim \mathbf{x}} [\ell_t(i)],$$

which capture the expected loss of choosing an expert from distribution $\mathbf{x} \in \Delta_n$ as a linear function.

The regret guarantees we have studied for OGD imply a regret of

$$O(GD\sqrt{T}) = O(\sqrt{nT}).$$

Here we have used the fact that the Eucliean diameter of the simplex is two, and that the losses are bounded by one, hence the Euclidean norm of the gradient vector ℓ_t is bounded by $\sqrt{n}$.

In contrast, the Hedge algorithm attains regret of $O(\sqrt{T \log n})$ for the same problem. How can we explain this discrepancy?!

5.2.1 The RFTL algorithm

Both OGD and Hedge are, in fact, instantiations of a more general meta-algorithm called RFTL (Regularized-Follow-The-Leader).

In an OCO setting of regret minimization, the most straightforward approach for the online player is to use at any time the optimal decision (i.e., point in the convex set) in hindsight. Formally, let

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x} \in \mathcal{K}} \sum_{\tau=1}^{t} f_\tau(\mathbf{x}).$$

This flavor of strategy is known as "fictitious play" in economics, and has been named "Follow the Leader" (FTL) in machine learning. It is not hard to see that this simple strategy fails miserably in a worst-case sense. That is, this strategy's regret can be linear in the number of iterations, as the following example shows: Consider $\mathcal{K} = [-1, 1]$, let $f_1(x) = \frac{1}{2}x$, and let f_τ for $\tau = 2, \ldots, T$ alternate between $-x$ or x. Thus,

$$\sum_{\tau=1}^{t} f_\tau(x) = \begin{cases} \frac{1}{2}x, & t \text{ is odd} \\ -\frac{1}{2}x, & \text{otherwise} \end{cases}$$

The FTL strategy will keep shifting between $x_t = -1$ and $x_t = 1$, always making the wrong choice.

The intuitive FTL strategy fails in the example above because it is unstable. Can we modify the FTL strategy such that it won't change decisions often, thereby causing it to attain low regret?

This question motivates the need for a general means of stabilizing the FTL method. Such a means is referred to as "regularization".

Algorithm 7 Regularized Follow The Leader

1: Input: $\eta > 0$, regularization function R, and a convex compact set $\mathcal{K}$.
2: Let $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{K}} \{R(\mathbf{x})\}$.
3: **for** $t = 1$ to T **do**
4: Predict $\mathbf{x}_t$.
5: Observe the payoff function f_t and let $\nabla_t = \nabla f_t(\mathbf{x}_t)$.
6: Update

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}\in\mathcal{K}} \left\{ \eta \sum_{s=1}^{t} \nabla_s^\top \mathbf{x} + R(\mathbf{x}) \right\}$$

7: **end for**

The generic RFTL meta-algorithm is defined in Algorithm 7. The regularization function R is assumed to be strongly convex, smooth, and twice differentiable.

5.2.2 Mirrored Descent

An alternative view of this algorithm is in terms of iterative updates, which can be spelled out using the above definition directly. The resulting algorithm is called "Mirrored Descent".

OMD is an iterative algorithm that computes the current decision using a simple gradient update rule and the previous decision, much like OGD. The generality of the method stems from the update being carried out in a "dual" space, where the duality notion is defined by the choice of regularization: the gradient of the regularization function defines a mapping from $\mathbb{R}^n$ onto itself, which is a vector field. The gradient updates are then carried out in this vector field.

For the RFTL algorithm the intuition was straightforward—the regularization was used to ensure stability of the decision. For OMD, regularization has an additional purpose: regularization transforms the space in which gradient updates are performed. This transformation enables better bounds in terms of the geometry of the space.

The OMD algorithm comes in two flavors: an agile and a lazy version. The lazy version keeps track of a point in Euclidean space and projects onto the convex decision set $\mathcal{K}$ only at decision time. In contrast, the agile version maintains a feasible point at all times, much like OGD.

Algorithm 8 Online Mirrored Descent

1: Input: parameter $\eta > 0$, regularization function $R(\mathbf{x})$.
2: Let $\mathbf{y}_1$ be such that $\nabla R(\mathbf{y}_1) = \mathbf{0}$ and $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}||\mathbf{y}_1)$.
3: **for** $t = 1$ to T **do**
4: Play $\mathbf{x}_t$.
5: Observe the payoff function f_t and let $\nabla_t = \nabla f_t(\mathbf{x}_t)$.
6: Update $\mathbf{y}_t$ according to the rule:

$$\begin{aligned} &\text{[Lazy version]} && \nabla R(\mathbf{y}_{t+1}) = \nabla R(\mathbf{y}_t) - \eta\,\nabla_t \\ &\text{[Agile version]} && \nabla R(\mathbf{y}_{t+1}) = \nabla R(\mathbf{x}_t) - \eta\,\nabla_t \end{aligned}$$

Project according to B_R:

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}||\mathbf{y}_{t+1})$$

7: **end for**

A myriad of questions arise, but first, let us see how does this algorithm give rise to both OGD.

We note that there are other important special cases of the RFTL meta-algorithm: those are derived with matrix-norm regularization—namely, the von Neumann entropy function, and the log-determinant function, as well as self-concordant barrier regularization. Perhaps most importantly for optimization, also the AdaGrad algorithm is obtained via changing regularization—which we shall explore in detail in the next chapter.

5.2.3 Deriving online gradient descent

To derive the online gradient descent algorithm, we take $R(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|_2^2$ for an arbitrary $\mathbf{x}_0 \in \mathcal{K}$. Projection with respect to this divergence is the standard Euclidean projection (left as an exercise), and in addition, $\nabla R(\mathbf{x}) = \mathbf{x} - \mathbf{x}_0$. Hence, the update rule for the OMD Algorithm 8 becomes:

$$\begin{aligned} &\mathbf{x}_t = \prod_{\mathcal{K}}(\mathbf{y}_t),\ \mathbf{y}_t = \mathbf{y}_{t-1} - \eta\nabla_{t-1} && \text{lazy version} \\ &\mathbf{x}_t = \prod_{\mathcal{K}}(\mathbf{y}_t),\ \mathbf{y}_t = \mathbf{x}_{t-1} - \eta\nabla_{t-1} && \text{agile version} \end{aligned}$$

The latter algorithm is exactly online gradient descent, as described in Algorithm 4 in Chapter 4. Furthermore, both variants are identical for the case in which $\mathcal{K}$ is the unit ball.

We later prove general regret bounds that will imply a $O(GD\sqrt{T})$ regret for OGD as a special case of mirrored descent.

5.2.4 Deriving multiplicative updates

Let $R(\mathbf{x}) = \mathbf{x}\log\mathbf{x} = \sum_i \mathbf{x}_i \log \mathbf{x}_i$ be the negative entropy function, where $\log\mathbf{x}$ is to be interpreted elementwise. Then $\nabla R(\mathbf{x}) = \mathbf{1} + \log\mathbf{x}$, and hence the update rules for the OMD algorithm become:

$$\mathbf{x}_t = \arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}||\mathbf{y}_t),\ \log\mathbf{y}_t = \log\mathbf{y}_{t-1} - \eta\nabla_{t-1} \qquad \text{lazy version}$$

$$\mathbf{x}_t = \arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}||\mathbf{y}_t),\ \log\mathbf{y}_t = \log\mathbf{x}_{t-1} - \eta\nabla_{t-1} \qquad \text{agile version}$$

With this choice of regularizer, a notable special case is the experts problem we encountered in §5.1, for which the decision set $\mathcal{K}$ is the n-dimensional simplex $\Delta_n = \{\mathbf{x} \in \mathbb{R}^n_+ \mid \sum_i \mathbf{x}_i = 1\}$. In this special case, the projection according to the negative entropy becomes scaling by the ℓ_1 norm (left as an exercise), which implies that both update rules amount to the same algorithm:

$$\mathbf{x}_{t+1}(i) = \frac{\mathbf{x}_t(i)\cdot e^{-\eta\nabla_t(i)}}{\sum_{j=1}^n \mathbf{x}_t(j)\cdot e^{-\eta\nabla_t(j)}},$$

which is exactly the Hedge algorithm! The general theorem we shall prove henceforth recovers the $O(\sqrt{T\log n})$ bound for prediction from expert advice for this algorithm.

5.3 Technical background: regularization functions

In the rest of this chapter we analyze the mirrored descent algorithm. For this purpose, consider regularization functions, denoted $R : \mathcal{K} \mapsto \mathbb{R}$, which are strongly convex and smooth (recall definitions in §2.1).

Although it is not strictly necessary, we assume that the regularization functions in this chapter are twice differentiable over $\mathcal{K}$ and, for all points $\mathbf{x} \in \text{int}(\mathcal{K})$ in the interior of the decision set, have a Hessian $\nabla^2 R(\mathbf{x})$ that is, by the strong convexity of R, positive definite.

We denote the diameter of the set $\mathcal{K}$ relative to the function R as

$$D_R = \sqrt{\max_{\mathbf{x},\mathbf{y}\in\mathcal{K}} \{R(\mathbf{x}) - R(\mathbf{y})\}}$$

Henceforth we make use of general norms and their dual. The dual norm to a norm $\|\cdot\|$ is given by the following definition:

$$\|\mathbf{y}\|^* \triangleq \max_{\|\mathbf{x}\|\leq 1} \langle \mathbf{x}, \mathbf{y} \rangle$$

A positive definite matrix A gives rise to the matrix norm $\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^\top A \mathbf{x}}$. The dual norm of a matrix norm is $\|\mathbf{x}\|_A^* = \|\mathbf{x}\|_{A^{-1}}$.

The generalized Cauchy-Schwarz theorem asserts $\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\| \|\mathbf{y}\|^*$ and in particular for matrix norms, $\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\|_A \|\mathbf{y}\|_A^*$.

In our derivations, we usually consider matrix norms with respect to $\nabla^2 R(\mathbf{x})$, the Hessian of the regularization function $R(\mathbf{x})$. In such cases, we use the notation

$$\|\mathbf{x}\|_{\mathbf{y}} \triangleq \|\mathbf{x}\|_{\nabla^2 R(\mathbf{y})}$$

and similarly

$$\|\mathbf{x}\|_{\mathbf{y}}^* \triangleq \|\mathbf{x}\|_{\nabla^{-2} R(\mathbf{y})}$$

A crucial quantity in the analysis with regularization is the remainder term of the Taylor approximation of the regularization function, and especially the remainder term of the first order Taylor approximation. The difference between the value of the regularization function at $\mathbf{x}$ and the value of the first order Taylor approximation is known as the Bregman divergence, given by

Definition 5.6. *Denote by $B_R(\mathbf{x}||\mathbf{y})$ the Bregman divergence with respect to the function R, defined as*

$$B_R(\mathbf{x}||\mathbf{y}) = R(\mathbf{x}) - R(\mathbf{y}) - \nabla R(\mathbf{y})^\top (\mathbf{x} - \mathbf{y})$$

For twice differentiable functions, Taylor expansion and the mean-value theorem assert that the Bregman divergence is equal to the second derivative at an intermediate point, i.e., (see exercises)

$$B_R(\mathbf{x}||\mathbf{y}) = \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|_{\mathbf{z}}^2,$$

for some point $\mathbf{z} \in [\mathbf{x}, \mathbf{y}]$, meaning there exists some $\alpha \in [0, 1]$ such that $\mathbf{z} = \alpha \mathbf{x} + (1 - \alpha)\mathbf{y}$. Therefore, the Bregman divergence defines a local norm, which has a dual norm. We shall denote this dual norm by

$$\|\cdot\|_{\mathbf{x},\mathbf{y}}^* \triangleq \|\cdot\|_{\mathbf{z}}^*.$$

With this notation we have

$$B_R(\mathbf{x}||\mathbf{y}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2_{\mathbf{x},\mathbf{y}}.$$

In online convex optimization, we commonly refer to the Bregman divergence between two consecutive decision points $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. In such cases, we shorthand notation for the norm defined by the Bregman divergence with respect to R on the intermediate point in $[\mathbf{x}_t, \mathbf{x}_{t+1}]$ as $\|\cdot\|_t \triangleq \|\cdot\|_{\mathbf{x}_t,\mathbf{x}_{t+1}}$. The latter norm is called the local norm at iteration t. With this notation, we have $B_R(\mathbf{x}_t||\mathbf{x}_{t+1}) = \frac{1}{2}\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_t^2$.

Finally, we consider below generalized projections that use the Bregman divergence as a distance instead of a norm. Formally, the projection of a point $\mathbf{y}$ according to the Bregman divergence with respect to function R is given by

$$\arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}||\mathbf{y})$$

5.4 Regret bounds for Mirrored Descent

In this subsection we prove regret bounds for the agile version of the RFTL algorithm. The analysis is quite different than the one for the lazy version, and of independent interest.

Theorem 5.7. *The RFTL Algorithm 8 attains for every $\mathbf{u} \in \mathcal{K}$ the following bound on the regret:*

$$\text{regret}_T \le 2\eta \sum_{t=1}^{T} \|\nabla_t\|_t^{*2} + \frac{R(\mathbf{u}) - R(\mathbf{x}_1)}{\eta}.$$

If an upper bound on the local norms is known, i.e. $\|\nabla_t\|_t^* \le G_R$ for all times t, then we can further optimize over the choice of η to obtain

$$\text{regret}_T \le 2D_R G_R \sqrt{2T}.$$

Proof. Since the functions $\mathbf{f}_t$ are convex, for any $\mathbf{x}^* \in K$,

$$\mathbf{f}_t(\mathbf{x}_t) - \mathbf{f}_t(\mathbf{x}^*) \le \nabla \mathbf{f}_t(\mathbf{x}_t)^\top (\mathbf{x}_t - \mathbf{x}^*).$$

The following property of Bregman divergences follows easily from the definition: for any vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$,

$$(\mathbf{x}-\mathbf{y})^\top(\nabla\mathcal{R}(\mathbf{z})-\nabla\mathcal{R}(\mathbf{y})) = B_\mathcal{R}(\mathbf{x},\mathbf{y}) - B_\mathcal{R}(\mathbf{x},\mathbf{z}) + B_\mathcal{R}(\mathbf{y},\mathbf{z}).$$

Combining both observations,

$$\begin{aligned}
2(\mathbf{f}_t(\mathbf{x}_t)-\mathbf{f}_t(\mathbf{x}^*)) &\le 2\nabla\mathbf{f}_t(\mathbf{x}_t)^\top(\mathbf{x}_t-\mathbf{x}^*) \\
&= \frac{1}{\eta}(\nabla\mathcal{R}(\mathbf{y}_{t+1})-\nabla\mathcal{R}(\mathbf{x}_t))^\top(\mathbf{x}^*-\mathbf{x}_t) \\
&= \frac{1}{\eta}[B_\mathcal{R}(\mathbf{x}^*,\mathbf{x}_t) - B_\mathcal{R}(\mathbf{x}^*,\mathbf{y}_{t+1}) + B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1})] \\
&\le \frac{1}{\eta}[B_\mathcal{R}(\mathbf{x}^*,\mathbf{x}_t) - B_\mathcal{R}(\mathbf{x}^*,\mathbf{x}_{t+1}) + B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1})]
\end{aligned}$$

where the last inequality follows from the generalized Pythagorean inequality (see [15] Lemma 11.3), as $\mathbf{x}_{t+1}$ is the projection w.r.t the Bregman divergence of $\mathbf{y}_{t+1}$ and $\mathbf{x}^* \in K$ is in the convex set. Summing over all iterations,

$$\begin{aligned}
2\text{regret} &\le \frac{1}{\eta}[B_\mathcal{R}(\mathbf{x}^*,\mathbf{x}_1) - B_\mathcal{R}(\mathbf{x}^*,\mathbf{x}_T)] + \sum_{t=1}^{T}\frac{1}{\eta}B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1}) \\
&\le \frac{1}{\eta}D^2 + \sum_{t=1}^{T}\frac{1}{\eta}B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1}) \qquad (5.1)
\end{aligned}$$

We proceed to bound $B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1})$. By definition of Bregman divergence, and the generalized Cauchy-Schwartz inequality,

$$\begin{aligned}
B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1}) + B_\mathcal{R}(\mathbf{y}_{t+1},\mathbf{x}_t) &= (\nabla\mathcal{R}(\mathbf{x}_t)-\nabla\mathcal{R}(\mathbf{y}_{t+1}))^\top(\mathbf{x}_t-\mathbf{y}_{t+1}) \\
&= \eta\nabla\mathbf{f}_t(\mathbf{x}_t)^\top(\mathbf{x}_t-\mathbf{y}_{t+1}) \\
&\le \eta\|\nabla\mathbf{f}_t(\mathbf{x}_t)\|^*\|\mathbf{x}_t-\mathbf{y}_{t+1}\| \\
&\le \frac{1}{2}\eta^2 G_*^2 + \frac{1}{2}\|\mathbf{x}_t-\mathbf{y}_{t+1}\|^2.
\end{aligned}$$

where in the last inequality follows from $(a-b)^2 \ge 0$. Thus, by our assumption $B_\mathcal{R}(\mathbf{x},\mathbf{y}) \ge \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$, we have

$$B_\mathcal{R}(\mathbf{x}_t,\mathbf{y}_{t+1}) \le \frac{1}{2}\eta^2 G_*^2 + \frac{1}{2}\|\mathbf{x}_t-\mathbf{y}_{t+1}\|^2 - B_\mathcal{R}(\mathbf{y}_{t+1},\mathbf{x}_t) \le \frac{1}{2}\eta^2 G_*^2.$$

Plugging back into Equation (5.1), and by non-negativity of the Bregman divergence, we get

$$\text{regret} \le \frac{1}{2}[\frac{1}{\eta}D^2 + \frac{1}{2}\eta T G_*^2] \le DG_*\sqrt{T}\ ,$$

by taking $\eta = \frac{D}{2\sqrt{T}G_*}$

□

5.5 Exercises

1. (a) Show that the dual norm to a matrix norm given by $A \succ 0$ corresponds to the matrix norm of A^{-1}.
   (b) Prove the generalized Cauchy-Schwarz inequality for any norm, i.e.,
   $$\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\| \|\mathbf{y}\|^*$$

2. Prove that the Bregman divergence is equal to the local norm at an intermediate point, that is:
   $$B_R(\mathbf{x}||\mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_{\mathbf{z}}^2,$$
   where $\mathbf{z} \in [\mathbf{x}, \mathbf{y}]$ and the interval $[\mathbf{x}, \mathbf{y}]$ is defined as
   $$[\mathbf{x}, \mathbf{y}] = \{\mathbf{v} = \alpha\mathbf{x} + (1-\alpha)\mathbf{y} \ , \ \alpha \in [0,1]\}$$

3. Let $R(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2$ be the (shifted) Euclidean regularization function. Prove that the corresponding Bregman divergence is the Euclidean metric. Conclude that projections with respect to this divergence are standard Euclidean projections.

4. Prove that both agile and lazy versions of the OMD meta-algorithm are equivalent in the case that the regularization is Euclidean and the decision set is the Euclidean ball.

5. For this problem the decision set is the n-dimensional simplex. Let $R(\mathbf{x}) = \mathbf{x} \log \mathbf{x}$ be the negative entropy regularization function. Prove that the corresponding Bregman divergence is the relative entropy, and prove that the diameter D_R of the n-dimensional simplex with respect to this function is bounded by $\log n$. Show that projections with respect to this divergence over the simplex amounts to scaling by the ℓ_1 norm.

6. * A set $\mathcal{K} \subseteq \mathbb{R}^d$ is symmetric if $\mathbf{x} \in \mathcal{K}$ implies $-\mathbf{x} \in \mathcal{K}$. Symmetric sets gives rise to a natural definition of a norm. Define the function $\|\cdot\|_{\mathcal{K}} : \mathbb{R}^d \mapsto \mathbb{R}$ as
   $$\|\mathbf{x}\|_{\mathcal{K}} = \arg\min_{\alpha > 0} \left\{ \frac{1}{\alpha}\mathbf{x} \in \mathcal{K} \right\}$$
   Prove that $\|\cdot\|_{\mathcal{K}}$ is a norm if and only if $\mathcal{K}$ is convex.

5.6 Bibliographic Remarks

Regularization in the context of online learning was first studied in [26] and [48]. The influential paper of Kalai and Vempala [45] coined the term "follow-the-leader" and introduced many of the techniques that followed in OCO. The latter paper studies random perturbation as a regularization and analyzes the follow-the-perturbed-leader algorithm, following an early development by [29] that was overlooked in learning for many years.

In the context of OCO, the term follow-the-regularized-leader was coined in [73, 71], and at roughly the same time an essentially identical algorithm was called "RFTL" in [1]. The equivalence of RFTL and Online Mirrored Descent was observed by [34].

Chapter 6

Adaptive Regularization

In the previous chapter we have studied a geometric extension of online / stochastic / determinisitic gradient descent. The technique to achieve it is called regularization, and we have seen how for the problem of prediction from expert advice, it can potentially given exponential improvements in the dependence on the dimension.

A natural question that arises is whether we can automatically learn the optimal regularization, i.e. best algorithm from the mirrored-descent class, for the problem at hand?

The answer is positive in a strong sense: it is theoretically possible to learn the optimal regularization online and in a data-specific way. Not only that, the resulting algorithms exhibit the most significant speedups in training deep neural networks from all accelerations studied thus far.

6.1 Adaptive Learning Rates: Intuition

The intuition for adaptive regularization is simple: consider an optimization problem which is axis-aligned, in which each coordinate is independent of the rest. It is reasonable to fine tune the learning rate for each coordinate separately - to achieve optimal convergence in that particular subspace of the problem, independently of the rest.

Thus, it is reasonable to change the SGD update rule from $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta \nabla_t$, to the more robust

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - D_t \nabla_t,$$

where D_t is a diagonal matrix that contains in coordinate (i, i) the learning rate for coordinate i in the gradient. Recall from the previous sections

that the optimal learning rate for stochastic non-convex optimization is of the order $O(\frac{1}{\sqrt{t}})$. More precisely, in Theorem 3.4, we have seen that this learning rate should be on the order of $O(\frac{1}{\sqrt{t\sigma^2}})$, where σ^2 is the variance of the stochastic gradients. The empirical estimator of the latter is $\sum_{i<t} \|\nabla_i\|^2$.

Thus, the robust version of stochastic gradient descent for smooth non-convex optimization should behave as the above equation, with

$$D_t(i,i) = \frac{1}{\sqrt{\sum_{i<t} \nabla_t(i)^2}}.$$

This is exactly the diagonal version of the AdaGrad algorithm! We continue to rigorously derive it and prove its performance guarantee.

6.2 A Regularization Viewpoint

In the previous chapter we have introduced regularization as a general methodology for deriving online convex optimization algorithms. Theorem 5.7 bounds the regret of the Mirrored Descent algorithm for any strongly convex regularizer as

$$\text{regret}_T \leq \max_{\mathbf{u}\in\mathcal{K}} \sqrt{2 \sum_t \|\nabla_t\|_t^{*2} B_R(\mathbf{u}||\mathbf{x}_1)}.$$

In addition, we have seen how to derive the online gradient descent and the multiplicative weights algorithms as special cases of the RFTL methodology.

We consider the following question: thus far we have thought of R as a strongly convex function. But which strongly convex function should we choose to minimize regret? This is a deep and difficult question which has been considered in the optimization literature since its early developments.

The ML approach is to learn the optimal regularization online. That is, a regularizer that adapts to the sequence of cost functions and is in a sense the "optimal" regularization to use in hindsight. We formalize this in the next section.

6.3 Tools from Matrix Calculus

Many of the inequalities that we are familiar with for positive real numbers hold for positive semi-definite matrices as well. We henceforth need the following inequality, which is left as an exercise,

Proposition 6.1. *For positive definite matrices $A \succcurlyeq B \succ 0$:*

$$2\mathbf{Tr}((A-B)^{1/2}) + \mathbf{Tr}(A^{-1/2}B) \leq 2\mathbf{Tr}(A^{1/2}).$$

Next, we require a structural result which explicitly gives the optimal regularization as a function of the gradients of the cost functions. For a proof see the exercises.

Proposition 6.2. *Let $A \succcurlyeq 0$. The minimizer of the following minimization problem:*

$$\begin{aligned} \min_{X} \quad & \mathbf{Tr}(X^{-1}A) \\ \textit{subject to } & X \succcurlyeq 0 \\ & \mathbf{Tr}(X) \leq 1, \end{aligned}$$

is $X = A^{1/2}/\mathbf{Tr}(A^{1/2})$, and the minimum objective value is $\mathbf{Tr}^2(A^{1/2})$.

6.4 The AdaGrad Algorithm and Its Analysis

To be more formal, let us consider the set of all strongly convex regularization functions with a fixed and bounded Hessian in the set

$$\forall \mathbf{x} \in \mathcal{K} \,.\, \nabla^2 R(\mathbf{x}) = \nabla^2 \in \mathcal{H} \triangleq \{X \in \mathbb{R}^{n \times n} \,;\, \mathbf{Tr}(X) \leq 1 \,,\, X \succcurlyeq 0\}$$

The set $\mathcal{H}$ is a restricted class of regularization functions (which does not include the entropic regularization). However, it is a general enough class to capture online gradient descent along with any rotation of the Euclidean regularization.

Algorithm 9 AdaGrad (Full Matrix version)

1: Input: parameters $\eta, \mathbf{x}_1 \in \mathcal{K}$.
2: Initialize: $S_0 = G_0 = \mathbf{0}$,
3: **for** $t = 1$ to T **do**
4: Predict $\mathbf{x}_t$, suffer loss $f_t(\mathbf{x}_t)$.
5: Update:

$$S_t = S_{t-1} + \nabla_t \nabla_t^\top, \; G_t = {S_t}^{1/2}$$

$$\mathbf{y}_{t+1} = \mathbf{x}_t - \eta G_t^{-1} \nabla_t$$

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x} \in \mathcal{K}} \|\mathbf{y}_{t+1} - \mathbf{x}\|_{G_t}^2$$

6: **end for**

The problem of learning the optimal regularization has given rise to Algorithm 9, known as the AdaGrad (Adaptive subGradient method) algorithm. In the algorithm definition and throughout this chapter, the notation A^{-1} refers to the Moore-Penrose pseudoinverse of the matrix A. Perhaps surprisingly, the regret of AdaGrad is at most a constant factor larger than the minimum regret of all RFTL algorithm with regularization functions whose Hessian is fixed and belongs to the class $\mathcal{H}$. The regret bound on AdaGrad is formally stated in the following theorem.

Theorem 6.3. *Let $\{\mathbf{x}_t\}$ be defined by Algorithm 9 with parameters $\eta = D$, where*

$$D = \max_{\mathbf{u} \in \mathcal{K}} \|\mathbf{u} - \mathbf{x}_1\|_2.$$

Then for any $\mathbf{x}^\star \in \mathcal{K}$,

$$\text{regret}_T(\textit{AdaGrad}) \leq 2D \sqrt{\min_{H \in \mathcal{H}} \sum_t \|\nabla_t\|_H^{*2}}.$$

Before proving this theorem, notice that it delivers on one of the promised accounts: comparing to the bound of Theorem 5.7 and ignoring the diameter D and dimensionality, the regret bound is as good as the regret of RFTL for the class of regularization functions.

We proceed to prove Theorem 6.3. First, a direct corollary of Proposition 6.2 is that

Corollary 6.4.

$$\begin{aligned}\sqrt{\min_{H \in \mathcal{H}} \sum_t \|\nabla_t\|_H^{*2}} &= \sqrt{\min_{H \in \mathcal{H}} \mathbf{Tr}(H^{-1} \textstyle\sum_t \nabla_t \nabla_t^\top)} \\ &= \mathbf{Tr} \sqrt{\textstyle\sum_t \nabla_t \nabla_t^\top} = \mathbf{Tr}(G_T)\end{aligned}$$

Hence, to prove Theorem 6.3, it suffices to prove the following lemma.

Lemma 6.5.

$$\text{regret}_T(\textit{AdaGrad}) \leq 2D\mathbf{Tr}(G_T) = 2D \sqrt{\min_{H \in \mathcal{H}} \sum_t \|\nabla_t\|_H^{*2}}.$$

Proof. By the definition of $\mathbf{y}_{t+1}$:

$$\mathbf{y}_{t+1} - \mathbf{x}^\star = \mathbf{x}_t - \mathbf{x}^\star - \eta {G_t}^{-1} \nabla_t, \tag{6.1}$$

and

$$G_t(\mathbf{y}_{t+1} - \mathbf{x}^\star) = G_t(\mathbf{x}_t - \mathbf{x}^\star) - \eta\nabla_t. \tag{6.2}$$

Multiplying the transpose of (6.1) by (6.2) we get

$$\begin{aligned}(\mathbf{y}_{t+1} - \mathbf{x}^\star)^\top G_t(\mathbf{y}_{t+1} - \mathbf{x}^\star) = \\ (\mathbf{x}_t - \mathbf{x}^\star)^\top G_t(\mathbf{x}_t - \mathbf{x}^\star) - 2\eta\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star) + \eta^2\nabla_t^\top G_t^{-1}\nabla_t.\end{aligned} \tag{6.3}$$

Since $\mathbf{x}_{t+1}$ is the projection of $\mathbf{y}_{t+1}$ in the norm induced by G_t, we have (see §2.1.1)

$$(\mathbf{y}_{t+1} - \mathbf{x}^\star)^\top G_t(\mathbf{y}_{t+1} - \mathbf{x}^\star) = \|\mathbf{y}_{t+1} - \mathbf{x}^\star\|_{G_t}^2 \geq \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|_{G_t}^2.$$

This inequality is the reason for using generalized projections as opposed to standard projections, which were used in the analysis of online gradient descent (see §4.4 Equation (4.2)). This fact together with (6.3) gives

$$\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star) \leq \frac{\eta}{2}\nabla_t^\top G_t^{-1}\nabla_t + \frac{1}{2\eta}\left(\|\mathbf{x}_t - \mathbf{x}^\star\|_{G_t}^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|_{G_t}^2\right).$$

Now, summing up over $t = 1$ to T we get that

$$\begin{aligned}\sum_{t=1}^T \nabla_t^\top(\mathbf{x}_t - \mathbf{x}^\star) &\leq \frac{\eta}{2}\sum_{t=1}^T \nabla_t^\top G_t^{-1}\nabla_t + \frac{1}{2\eta}\|\mathbf{x}_1 - \mathbf{x}^\star\|_{G_0}^2 \\ &+ \frac{1}{2\eta}\sum_{t=1}^T\left(\|\mathbf{x}_t - \mathbf{x}^\star\|_{G_t}^2 - \|\mathbf{x}_t - \mathbf{x}^\star\|_{G_{t-1}}^2\right) - \frac{1}{2\eta}\|\mathbf{x}_{T+1} - \mathbf{x}^\star\|_{G_T}^2 \\ &\leq \frac{\eta}{2}\sum_{t=1}^T \nabla_t^\top G_t^{-1}\nabla_t + \frac{1}{2\eta}\sum_{t=1}^T(\mathbf{x}_t - \mathbf{x}^\star)^\top(G_t - G_{t-1})(\mathbf{x}_t - \mathbf{x}^\star).\end{aligned} \tag{6.4}$$

In the last inequality we use the fact that $G_0 = \mathbf{0}$. We proceed to bound each of the terms above separately.

Lemma 6.6. *With S_t, G_t as defined in Algorithm 9,*

$$\sum_{t=1}^T \nabla_t^\top G_t^{-1}\nabla_t \leq 2\sum_{t=1}^T \nabla_t^\top G_T^{-1}\nabla_t \leq 2\mathbf{Tr}(G_T).$$

Proof. We prove the lemma by induction. The base case follows since

$$\begin{aligned}\nabla_1^\top G_1^{-1}\nabla_1 &= \mathbf{Tr}(G_1^{-1}\nabla_1\nabla_1^\top) \\ &= \mathbf{Tr}(G_1^{-1}G_1^2) \\ &= \mathbf{Tr}(G_1).\end{aligned}$$

Assuming the lemma holds for $T-1$, we get by the inductive hypothesis

$$\begin{aligned}\sum_{t=1}^{T} \nabla_t^\top G_t^{-1} \nabla_t &\leq 2\mathbf{Tr}(G_{T-1}) + \nabla_T^\top G_T^{-1} \nabla_T \\ &= 2\mathbf{Tr}((G_T^2 - \nabla_T \nabla_T^\top)^{1/2}) + \mathbf{Tr}(G_T^{-1} \nabla_T \nabla_T^\top) \\ &\leq 2\mathbf{Tr}(G_T).\end{aligned}$$

Here, the last inequality is due to the matrix inequality 6.1. □

Lemma 6.7.

$$\sum_{t=1}^{T} (\mathbf{x}_t - \mathbf{x}^\star)^\top (G_t - G_{t-1})(\mathbf{x}_t - \mathbf{x}^\star) \leq D^2 \mathbf{Tr}(G_T).$$

Proof. By definition $S_t \succcurlyeq S_{t-1}$, and hence $G_t \succcurlyeq G_{t-1}$. Thus,

$$\begin{aligned}&\sum_{t=1}^{T} (\mathbf{x}_t - \mathbf{x}^\star)^\top (G_t - G_{t-1})(\mathbf{x}_t - \mathbf{x}^\star) \\ &\leq \sum_{t=1}^{T} D^2 \lambda_{\max}(G_t - G_{t-1}) \\ &\leq D^2 \sum_{t=1}^{T} \mathbf{Tr}(G_t - G_{t-1}) && A \succcurlyeq 0 \;\Rightarrow\; \lambda_{\max}(A) \leq \mathbf{Tr}(A) \\ &= D^2 \sum_{t=1}^{T} (\mathbf{Tr}(G_t) - \mathbf{Tr}(G_{t-1})) && \text{linearity of the trace} \\ &\leq D^2 \mathbf{Tr}(G_T).\end{aligned}$$

□

Plugging both lemmas into Equation (6.4), we obtain

$$\sum_{t=1}^{T} \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^\star) \leq \; \eta \mathbf{Tr}(G_T) + \frac{1}{2\eta} D^2 \mathbf{Tr}(G_T) \leq 2D\mathbf{Tr}(G_T).$$

□

6.5 Diagonal AdaGrad

The AdaGrad algorithm maintains potentially dense matrices, and requires the computation of the square root of these matrices. This is usually prohibitive in machine learning applications in which the dimension is very large. Fortunately, the same ideas can be applied with almost no computational overhead on top of vanilla SGD, using the diagonal version of AdaGrad given by:

Algorithm 10 AdaGrad (diagonal version)

1: Input: parameters $\eta, \mathbf{x}_1 \in \mathcal{K}$.
2: Initialize: $S_0 = G_0 = \mathbf{0}$,
3: **for** $t = 1$ to T **do**
4: Predict $\mathbf{x}_t$, suffer loss $f_t(\mathbf{x}_t)$.
5: Update:

$$S_t = S_{t-1} + \mathrm{diag}(\nabla_t \nabla_t^\top),\ G_t = {S_t}^{1/2}$$

$$\mathbf{y}_{t+1} = \mathbf{x}_t - \eta G_t^{-1} \nabla_t$$

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x} \in \mathcal{K}} \|\mathbf{y}_{t+1} - \mathbf{x}\|_{G_t}^2$$

6: **end for**

In contrast to the full-matrix version, this version can be implemented in linear time and space, since diagonal matrices can be manipulated as vectors. Thus, memory overhead is only a single d-dimensional vector, which is used to represent the diagonal preconditioning (regularization) matrix, and the computational overhead is a few vector manipulations per iteration.

Very similar to the full matrix case, the diagonal AdaGrad algorithm can be analyzed and the following performance bound obtained:

Theorem 6.8. *Let $\{\mathbf{x}_t\}$ be defined by Algorithm 10 with parameters $\eta = D_\infty$, where*

$$D_\infty = \max_{\mathbf{u} \in \mathcal{K}} \|\mathbf{u} - \mathbf{x}_1\|_\infty,$$

and let diag($\mathcal{H}$) be the set of all diagonal matrices in $\mathcal{H}$. Then for any $\mathbf{x}^\star \in \mathcal{K}$,

$$\mathrm{regret}_T(\textit{D-AdaGrad}) \leq 2 D_\infty \sqrt{\min_{H \in \textit{diag}(\mathcal{H})} \sum_t \|\nabla_t\|_H^{*2}}.$$

6.6 State-of-the-art: from Adam to Shampoo and beyond

Since the introduction of the adaptive regularization technique in the context of regret minimization, several improvements were introduced that now compose state-of-the-art. A few notable advancements include:

AdaDelta: The algorithm keeps an exponential average of past gradients and uses that in the update step.

Adam: Adds a sliding window to AdaGrad, as well as adding a form of momentum via estimating the second moments of past gradients and adjusting the update accordingly.

Shampoo: Interpolates between full-matrix and diagonal adagrad in the context of deep neural networks: use of the special layer structure to reduce memory constraints.

AdaFactor: Suggests a Shampoo-like approach to reduce memory footprint even further, to allow the training of huge models.

GGT: While full-matrix AdaGrad is computationally slow due to the cost of manipulating matrices, this algorithm uses recent gradients (a thin matrix G), and via linear algebraic manipulations reduces computation by never computing $GG^\top$, but rather only $G^\top G$, which is low dimensional.

SM3 , ET: Diagonal AdaGrad requires an extra $O(n)$ memory to store $\text{diag}(G_t)$. These algorithms, inspired by AdaFactor, approximate G_t as a low rank tensor to save memory and computation.

6.7 Exercises

1. * Prove that for positive definite matrices $A \succcurlyeq B \succ 0$ it holds that
   (a) $A^{1/2} \succcurlyeq B^{1/2}$
   (b) $2\mathbf{Tr}((A-B)^{1/2}) + \mathbf{Tr}(A^{-1/2}B) \leq 2\mathbf{Tr}(A^{1/2})$.

2. * Consider the following minimization problem where $A \succ 0$:

$$\begin{aligned} \min_X \quad & \mathbf{Tr}(X^{-1}A) \\ \text{subject to} \quad & X \succ 0 \\ & \mathbf{Tr}(X) \leq 1. \end{aligned}$$

   Prove that its minimizer is given by $X = A^{1/2}/\mathbf{Tr}(A^{1/2})$, and the minimum is obtained at $\mathbf{Tr}^2(A^{1/2})$.

6.8 Bibliographic Remarks

The AdaGrad algorithm was introduced in [19, 18], its diagonal version was also discovered in parallel in [52]. Adam [46] and RMSprop [39] are widely used methods based on adaptive regularization. A cleaner analysis was recently proposed in [27], see also [17].

Adaptive regularization has received much attention recently, see e.g., [60, 85]. Newer algorithmic developments on adaptive regularization include Shampoo [28], GGT [3], AdaFactor [77], Extreme Tensoring [16] and SM3 [6].

Chapter 7

Variance Reduction

In the previous chapter we have studied the first of our three acceleration techniques over SGD, adaptive regularization, which is a geometric tool for acceleration. In this chapter we introduce the second first-order acceleration technique, called variance reduction. This technique is probabilistic in nature, and applies to more restricted settings of mathematical optimization in which the objective function has a finite-sum structure. Namely, we consider optimization problems of the form

$$\min_{\mathbf{x}\in\mathcal{K}} f(\mathbf{x}) \ , \ f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x}) \ . \tag{7.1}$$

Such optimization problems are canonical in training of ML models, convex and non-convex. However, in the context of machine learning we should remember that the ultimate goal is generalization rather than training.

7.1 Variance reduction: Intuition

The intuition for variance reduction is simple, and comes from trying to improve the naive convergence bounds for SGD that we have covered in the first lesson.

Recall the SGD update rule $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta\hat{\nabla}_t$, in which $\hat{\nabla}_t$ is an unbiased estimator for the gradient such that

$$\mathbf{E}[\hat{\nabla}_t] = \nabla_t \ , \ \mathbf{E}[\|\hat{\nabla}_t\|_2^2] \leq \sigma^2.$$

We have seen in Theorem 3.4, that for this update rule,

$$\mathbf{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq 2\sqrt{\frac{M\beta\sigma^2}{T}}.$$

The convergence is proportional to the second moment of the gradient estimator, and thus it makes sense to try to reduce this second moment. The variance reduction technique attempts to do so by using the average of all previous gradients, as we show next.

7.2 Setting and definitions

We consider the ERM optimization problem over an average of loss functions. Before we begin, we need a few preliminaries and assumptions:

1. We denote distance to optimality according to function value as

$$h_t = f(\mathbf{x}_t) - f(\mathbf{x}^*),$$

   and in the k'th epoch of an algorithm, we denote $h_t^k = f(\mathbf{x}_t^k) - f(\mathbf{x}^*)$.

2. We denote $\tilde{h}_k = \max\left\{4h_0^k \ , \ 8\alpha D_k^2\right\}$ over an epoch.

3. Assume all stochastic gradients have bounded second moments

$$\|\hat{\nabla}_t\|_2^2 \leq \sigma^2.$$

4. We will assume that the individual functions f_i in formulation (7.1) are also $\hat{\beta}$-smooth and have $\hat{\beta}$-Lipschitz gradient, namely

$$\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq \hat{\beta}\|\mathbf{x} - \mathbf{y}\|.$$

5. We will use, proved in Lemma 2.3, that for β-smooth and α-strongly convex f we have

$$h_t \geq \frac{1}{2\beta}\|\nabla_t\|^2$$

   and

$$\frac{\alpha}{2}d_t^2 = \frac{\alpha}{2}\|\mathbf{x}_t - \mathbf{x}^*\|^2 \leq h_t \leq \frac{1}{2\alpha}\|\nabla_t\|^2.$$

6. Recall that a function f is γ-well-conditioned if it is β-smooth, α-strongly convex and $\gamma \leq \frac{\alpha}{\beta}$.

7.3 The variance reduction advantage

Consider gradient descent for γ-well conditioned functions, and specifically used for ML training as in formulation (7.1) . It is well known that GD attains linear convergence rate as we now prove for completeness:

Theorem 7.1. *For unconstrained minimization of γ-well-conditioned functions and $\eta_t = \frac{1}{\beta}$, the Gradient Descent Algorithm 2 converges as*

$$h_{t+1} \leq h_1 e^{-\gamma t}.$$

Proof.

$$\begin{aligned}
h_{t+1} - h_t &= f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) \\
&\leq \nabla_t^\top (\mathbf{x}_{t+1} - \mathbf{x}_t) + \frac{\beta}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 && \beta\text{-smoothness} \\
&= -\eta_t \|\nabla_t\|^2 + \frac{\beta}{2}\eta_t^2 \|\nabla_t\|^2 && \text{algorithm defn.} \\
&= -\frac{1}{2\beta}\|\nabla_t\|^2 && \text{choice of } \eta_t = \frac{1}{\beta} \\
&\leq -\frac{\alpha}{\beta} h_t. && \text{by (2.1)}
\end{aligned}$$

Thus,

$$h_{t+1} \leq h_t(1 - \frac{\alpha}{\beta}) \leq \cdots \leq h_1(1-\gamma)^t \leq h_1 e^{-\gamma t}$$

where the last inequality follows from $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$. □

However, what is the overall computational cost? Assuming that we can compute the gradient of each loss function corresponding to the individual training examples in $O(d)$ time, the overall running time to compute the gradient is $O(md)$.

In order to attain approximation ε to the objective, the algorithm requires $O(\frac{1}{\gamma}\log\frac{1}{\varepsilon})$ iterations, as per the Theorem above. Thus, the overall running time becomes $O(\frac{md}{\gamma}\log\frac{1}{\varepsilon})$. As we show below, variance reduction can reduce this running time to be $O((m + \frac{1}{\tilde{\gamma}^2})d\log\frac{1}{\varepsilon})$, where $\tilde{\gamma}$ is a different condition number for the same problem, that is in general smaller than the original. Thus, in one line, the variance reduction advantage can be summarized as:

$$\boxed{\frac{md}{\gamma}\log\frac{1}{\varepsilon} \mapsto (m + \frac{1}{\tilde{\gamma}^2})d\log\frac{1}{\varepsilon}\ .}$$

7.4 A simple variance-reduced algorithm

The following simple variance-reduced algorithm illustrates the main ideas of the technique. The algorithm is a stochastic gradient descent variant which proceeds in epochs. Strong convexity implies that the distance to the optimum shrinks with function value, so it is safe to decrease the distance upper bound every epoch.

The main innovation is in line 7, which constructs the gradient estimator. Instead of the usual trick - which is to sample one example at random - here the estimator uses the entire gradient computed at the beginning of the current epoch.

Algorithm 11 Epoch GD

1: Input: f, T, $\mathbf{x}_0^1 \in \mathcal{K}$, upper bound $D_1 \geq \|\mathbf{x}_0^1 - \mathbf{x}^*\|$, step sizes $\{\eta_t\}$
2: **for** $k = 1$ to $\log \frac{1}{\varepsilon}$ **do**
3: Let $B_{D_k}(\mathbf{x}_0^k)$ be the ball of radius D_k around $\mathbf{x}_0^k$.
4: compute full gradient $\nabla_0^k = \nabla f(\mathbf{x}_0^k)$
5: **for** $t = 1$ to T **do**
6: Sample $i_t \in [m]$ uniformly at random, let $f_t = f_{i_t}$.
7: construct stochastic gradient $\hat{\nabla}_t^k = \nabla f_t(\mathbf{x}_t^k) - \nabla f_t(\mathbf{x}_0^k) + \nabla_0^k$
8: Let $\mathbf{y}_{t+1}^k = \mathbf{x}_t^k - \eta_t \hat{\nabla}_t^k$, $\mathbf{x}_{t+1} = \Pi_{B_{D_k}(\mathbf{x}_0^k)}(\mathbf{y}_{t+1})$
9: **end for**
10: Set $\mathbf{x}_0^{k+1} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t^k$. $D_{k+1} \leftarrow D_k/2$.
11: **end for**
12: **return** $\mathbf{x}_{T+1}^0$

The main guarantee for this algorithm is the following theorem, which delivers upon the aforementioned improvement,

Theorem 7.2. *Algorithm 11 returns an ε-approximate solution to optimization problem* (7.1) *in total time*

$$O\left(\left(m + \frac{1}{\tilde{\gamma}^2}\right) d \log \frac{1}{\varepsilon}\right).$$

Let $\tilde{\gamma} = \frac{\alpha}{\hat{\beta}} < \gamma$. Then the proof of this theorem follows from the following lemma.

Lemma 7.3. *For $T = \tilde{O}\left(\frac{1}{\tilde{\gamma}^2}\right)$, we have*

$$\mathbf{E}[\tilde{h}_{k+1}] \leq \frac{1}{2}\tilde{h}_k.$$

Proof. As a first step, we bound the variance of the gradients. Due to the fact that $\mathbf{x}_t^k \in B_{D_k}(\mathbf{x}_0^k)$, we have that for $k' > k$, $\|\mathbf{x}_t^k - \mathbf{x}_t^{k'}\|^2 \leq 4D_k^2$. Thus,

$$\begin{aligned}
\|\hat{\nabla}_t^k\|^2 &= \|\nabla f_t(\mathbf{x}_t^k) - \nabla f_t(\mathbf{x}_0^k) + \nabla f(\mathbf{x}_0^k)\|^2 && \text{definition} \\
&\leq 2\|\nabla f_t(\mathbf{x}_t^k) - \nabla f_t(\mathbf{x}_0^k)\|^2 + 2\|\nabla f(\mathbf{x}_0^k)\|^2 && (a+b)^2 \leq 2a^2 + 2b^2 \\
&\leq 2\hat{\beta}^2\|\mathbf{x}_t^k - \mathbf{x}_0^k\|^2 + 4\beta h_0^k && \text{smoothness} \\
&\leq 8\hat{\beta}^2 D_k^2 + 4\beta h_0^k && \text{projection step} \\
&\leq \hat{\beta}^2 \tfrac{1}{\alpha}\tilde{h}_k + 4\beta h_0^k \leq \tilde{h}_k(\tfrac{\hat{\beta}^2}{\alpha} + \beta)
\end{aligned}$$

Next, using the regret bound for strongly convex functions, we have

$$\begin{aligned}
\mathbf{E}[h_0^{k+1}] &\leq \mathbf{E}[\tfrac{1}{T}\textstyle\sum_t h_t^k] && \text{Jensen} \\
&\leq \tfrac{1}{\alpha T}\,\mathbf{E}[\textstyle\sum_t \frac{1}{t}\|\hat{\nabla}_t^k\|^2] && \text{Theorem 4.4} \\
&\leq \tfrac{1}{\alpha T}\textstyle\sum_t \frac{1}{t}\tilde{h}_k(\frac{\hat{\beta}^2}{\alpha} + \beta) && \text{above} \\
&\leq \tfrac{\log T}{T}\tilde{h}_k(\tfrac{1}{\tilde{\gamma}^2} + \tfrac{1}{\gamma}) && \tilde{\gamma} = \frac{\alpha}{\hat{\beta}}
\end{aligned}$$

Which implies the Lemma by choice of T, definition of $\tilde{h}_k = \max\left\{4h_0^k\ ,\ 8\alpha D_k^2\right\}$, and exponential decrease of D_k.

The expectation is over the stochastic gradient definition, and is required for using Theorem 4.4. □

To obtain the theorem from the lemma above, we need to strengthen it to a high probability statement using a martingale argument. This is possible since the randomness in construction of the stochastic gradients is i.i.d.

The lemma now implies the theorem by noting that $O(\log \frac{1}{\varepsilon})$ epochs suffices to get ε-approximation. Each epoch requires the computation of one full gradient, in time $O(md)$, and $\tilde{O}(\frac{1}{\tilde{\gamma}^2})$ iterations that require stochastic gradient computation, in time $O(d)$.

7.5 Bibliographic Remarks

The variance reduction technique was first introduced as part of the SAG algorithm [70]. Since then a host of algorithms were developed using the technique. The simplest exposition of the technique was given in [44]. The exposition in this chapter is developed from the Epoch GD algorithm [37], which uses a related technique for stochastic strongly convex optimization, as developed in [86].

Chapter 8

Nesterov Acceleration

In previous chapters we have studied our bread and butter technique, SGD, as well as two acceleration techniques of adaptive regularization and variance reduction. In this chapter we study the historically earliest acceleration technique, known as Nesterov acceleration, or simply "acceleration".

For smooth and convex functions, Nesterov acceleration improves the convergence rate to optimality to $O(\frac{1}{T^2})$, a quadratic improvement over vanilla gradient descent. Similar accelerations are possible when the function is also strongly convex: an accelerated rate of $e^{-\sqrt{\gamma}T}$, where γ is the condition number, vs. $e^{-\gamma T}$ of vanilla gradient descent. This improvement is theoretically very significant.

However, in terms of applicability, Nesterov acceleration is theoretically the most restricted in the context of machine learning: it requires a smooth and convex objective. More importantly, the learning rates of this method are very brittle, and the method is not robust to noise. Since noise is predominant in machine learning, the theoretical guarantees in stochastic optimization environments are very restricted.

However, the heuristic of momentum, which historically inspired acceleration, is extremely useful for non-convex stochastic optimization (although not known to yield significant improvements in theory).

8.1 Algorithm and implementation

Nesterov acceleration applies to the general setting of constrained smooth convex optimization:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}). \tag{8.1}$$

For simplicity of presentation, we restrict ourselves to the unconstrained convex and smooth case. Nevertheless, the method can be extended to constrained smooth convex, and potentially strongly convex, settings.

The simple method presented in Algorithm 12 below is computationally equivalent to gradient descent. The only overhead is saving three state vectors (that can be reduced to two) instead of one for gradient descent. The following simple accelerated algorithm illustrates the main ideas of the technique.

Algorithm 12 Simplified Nesterov Acceleration

Input: f, T, initial point $\mathbf{x}_0$, parameters η, β, τ.
for $t = 1$ to T **do**
  Set $\mathbf{x}_{t+1} = \tau \mathbf{z}_t + (1-\tau)\mathbf{y}_t$, and denote $\nabla_{t+1} = \nabla f(\mathbf{x}_{t+1})$.
  Let $\mathbf{y}_{t+1} = \mathbf{x}_{t+1} - \frac{1}{\beta}\nabla_{t+1}$
  Let $\mathbf{z}_{t+1} = \mathbf{z}_t - \eta \nabla_{t+1}$
end for
return $\bar{\mathbf{x}} = \frac{1}{T}\sum_t \mathbf{x}_t$

8.2 Analysis

The main guarantee for this algorithm is the following theorem.

Theorem 8.1. *Algorithm 12 converges to an ε-approximate solution to optimization problem* (8.1) *in $O(\frac{1}{\sqrt{\varepsilon}})$ iterations.*

The proof starts with the following lemma which follows from our earlier standard derivations.

Lemma 8.2.

$$\eta \nabla_{t+1}^\top (\mathbf{z}_t - \mathbf{x}^*) \le 2\eta^2 \beta (f(\mathbf{x}_{t+1}) - f(\mathbf{y}_{t+1})) + \left[\|\mathbf{z}_t - \mathbf{x}^*\|^2 - \|\mathbf{z}_{t+1} - \mathbf{x}^*\|^2\right].$$

Proof. The proof is very similar to that of Theorem 4.2. By definition of $\mathbf{z}_t$,[1]

$$\begin{aligned}
\|\mathbf{z}_{t+1} - \mathbf{x}^*\|^2 &= \|\mathbf{z}_t - \eta\nabla_{t+1} - \mathbf{x}^*\|^2 \\
&= \|\mathbf{z}_t - \mathbf{x}^*\|^2 - \eta\nabla_{t+1}^\top(\mathbf{z}_t - \mathbf{x}^*) + \eta^2\|\nabla_{t+1}\|^2 \\
&\le \|\mathbf{z}_t - \mathbf{x}^*\|^2 - \eta\nabla_{t+1}^\top(\mathbf{z}_t - \mathbf{x}^*) + 2\eta^2\beta(f(\mathbf{x}_{t+1}) - f(\mathbf{y}_{t+1})) \quad \text{Lemma 2.3 part 3}
\end{aligned}$$

□

[1] Henceforth we use Lemma 2.3 part 3. This proof of this Lemma shows that for $\mathbf{y} = \mathbf{x} - \frac{1}{\beta}\nabla f(\mathbf{x})$, it holds that $f(\mathbf{x}) - f(\mathbf{y}) \ge \frac{1}{2\beta}\|\nabla f(\mathbf{x})\|^2$.

Lemma 8.3. *For $2\eta\beta = \frac{1-\tau}{\tau}$, we have that*

$$\eta\nabla_{t+1}^{\top}(\mathbf{x}_{t+1} - \mathbf{x}^*) \leq 2\eta^2\beta(f(\mathbf{y}_t) - f(\mathbf{y}_{t+1})) + \left[\|\mathbf{z}_t - \mathbf{x}^*\|^2 - \|\mathbf{z}_{t+1} - \mathbf{x}^*\|^2\right].$$

Proof.

$$\begin{aligned}
&\eta\nabla_{t+1}^{\top}(\mathbf{x}_{t+1} - \mathbf{x}^*) - \eta\nabla_{t+1}^{\top}(\mathbf{z}_t - \mathbf{x}^*) \\
&\qquad = \eta\nabla_{t+1}^{\top}(\mathbf{x}_{t+1} - \mathbf{z}_t) \\
&\qquad = \tfrac{(1-\tau)\eta}{\tau}\nabla_{t+1}^{\top}(\mathbf{y}_t - \mathbf{x}_{t+1}) && \tau(\mathbf{x}_{t+1} - \mathbf{z}_t) = (1-\tau)(\mathbf{y}_t - \mathbf{x}_{t+1}) \\
&\qquad \leq \tfrac{(1-\tau)\eta}{\tau}(f(\mathbf{y}_t) - f(\mathbf{x}_{t+1})). && \text{convexity}
\end{aligned}$$

Thus, in combination with Lemma 8.2, and the condition of the Lemma, we get the inequality. □

We can now sketch the proof of the main theorem.

Proof. Telescope Lemma 8.3 for all iterations to obtain:

$$\begin{aligned}
Th_T &= T(f(\bar{\mathbf{x}}) - f(\mathbf{x}^*)) \\
&\leq \textstyle\sum_t \nabla_t^{\top}(\mathbf{x}_t - \mathbf{x}^*) \\
&\leq 2\eta\beta\textstyle\sum_t(f(\mathbf{y}_t) - f(\mathbf{y}_{t+1})) + \frac{1}{\eta}\sum_t\left[\|\mathbf{z}_t - \mathbf{x}^*\|^2 - \|\mathbf{z}_{t+1} - \mathbf{x}^*\|^2\right] \\
&\leq 2\eta\beta(f(\mathbf{y}_1) - f(\mathbf{y}_{T+1})) + \tfrac{1}{\eta}\left[\|\mathbf{z}_1 - \mathbf{x}^*\|^2 - \|\mathbf{z}_{T+1} - \mathbf{x}^*\|^2\right] \\
&\leq \sqrt{2\beta h_1 D}, && \text{optimizing } \eta
\end{aligned}$$

where h_1 is an upper bound on the distance $f(\mathbf{y}_1) - f(\mathbf{x}^*)$, and D bounds the Euclidean distance of $\mathbf{z}_t$ to the optimum. Thus, we get a recurrence of the form

$$h_T \leq \frac{\sqrt{h_1}}{T}.$$

Restarting Algorithm 12 and adapting the learning rate according to h_T gives a rate of convergence of $O(\frac{1}{T^2})$ to optimality. □

8.3 Bibliographic Remarks

Accelerated rates of order $O(\frac{1}{T^2})$ were obtained by Nemirovski as early as the late seventies. The first practically efficient accelerated algorithm is due to Nesterov [56] , see also [57]. The simplified proof presented hereby is due to [5].

Chapter 9

The conditional gradient method

In many computational and learning scenarios the main bottleneck of optimization, both online and offline, is the computation of projections onto the underlying decision set (see §2.1.1). In this chapter we discuss projection-free methods in convex optimization, and some of their applications in machine learning.

The motivating example throughout this chapter is the problem of matrix completion, which is a widely used and accepted model in the construction of recommendation systems. For matrix completion and related problems, projections amount to expensive linear algebraic operations and avoiding them is crucial in big data applications.

Henceforth we describe the conditional gradient algorithm, also known as the Frank-Wolfe algorithm. Afterwards, we describe problems for which linear optimization can be carried out much more efficiently than projections. We conclude with an application to exploration in reinforcement learning.

9.1 Review: relevant concepts from linear algebra

This chapter addresses rectangular matrices, which model applications such as recommendation systems naturally. Consider a matrix $X \in \mathbb{R}^{n \times m}$. A non-negative number $\sigma \in \mathbb{R}_+$ is said to be a singular value for X if there are two vectors $\mathbf{u} \in \mathbb{R}^n, \mathbf{v} \in \mathbb{R}^m$ such that

$$X^\top \mathbf{u} = \sigma \mathbf{v}, \quad X\mathbf{v} = \sigma \mathbf{u}.$$

The vectors $\mathbf{u}, \mathbf{v}$ are called the left and right singular vectors respectively. The non-zero singular values are the square roots of the eigenvalues of the matrix $XX^\top$ (and $X^\top X$). The matrix X can be written as

$$X = U\Sigma V^\top \ , \ U \in \mathbb{R}^{n\times\rho} \ , \ V^\top \in \mathbb{R}^{\rho\times m},$$

where $\rho = \min\{n, m\}$, the matrix U is an orthogonal basis of the left singular vectors of X, the matrix V is an orthogonal basis of right singular vectors, and Σ is a diagonal matrix of singular values. This form is called the singular value decomposition for X.

The number of non-zero singular values for X is called its rank, which we denote by $k \leq \rho$. The nuclear norm of X is defined as the ℓ_1 norm of its singular values, and denoted by

$$\|X\|_* = \sum_{i=1}^{\rho} \sigma_i$$

It can be shown (see exercises) that the nuclear norm is equal to the trace of the square root of the matrix times its transpose, i.e.,

$$\|X\|_* = \mathbf{Tr}(\sqrt{X^\top X})$$

We denote by $A \bullet B$ the inner product of two matrices as vectors in $\mathbb{R}^{n\times m}$, that is

$$A \bullet B = \sum_{i=1}^{n}\sum_{j=1}^{m} A_{ij}B_{ij} = \mathbf{Tr}(AB^\top)$$

9.2 Motivation: matrix completion and recommendation systems

Media recommendations have changed significantly with the advent of the Internet and rise of online media stores. The large amounts of data collected allow for efficient clustering and accurate prediction of users' preferences for a variety of media. A well-known example is the so called "Netflix challenge"—a competition of automated tools for recommendation from a large dataset of users' motion picture preferences.

One of the most successful approaches for automated recommendation systems, as proven in the Netflix competition, is matrix completion. Perhaps the simplest version of the problem can be described as follows.

The entire dataset of user-media preference pairs is thought of as a partially-observed matrix. Thus, every person is represented by a row in

the matrix, and every column represents a media item (movie). For simplicity, let us think of the observations as binary—a person either likes or dislikes a particular movie. Thus, we have a matrix $M \in \{0, 1, *\}^{n \times m}$ where n is the number of persons considered, m is the number of movies at our library, and $0/1$ and $*$ signify "dislike", "like" and "unknown" respectively:

$$M_{ij} = \begin{cases} 0, & \text{person } i \text{ dislikes movie } j \\ 1, & \text{person } i \text{ likes movie } j \\ *, & \text{preference unknown} \end{cases}.$$

The natural goal is to complete the matrix, i.e. correctly assign 0 or 1 to the unknown entries. As defined so far, the problem is ill-posed, since any completion would be equally good (or bad), and no restrictions have been placed on the completions.

The common restriction on completions is that the "true" matrix has low rank. Recall that a matrix $X \in \mathbb{R}^{n \times m}$ has rank $k < \rho = \min\{n, m\}$ if and only if it can be written as

$$X = UV \ , \ U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{k \times m}.$$

The intuitive interpretation of this property is that each entry in M can be explained by only k numbers. In matrix completion this means, intuitively, that there are only k factors that determine a persons preference over movies, such as genre, director, actors and so on.

Now the simplistic matrix completion problem can be well-formulated as in the following mathematical program. Denote by $\|\cdot\|_{OB}$ the Euclidean norm only on the observed (non starred) entries of M, i.e.,

$$\|X\|_{OB}^2 = \sum_{M_{ij} \neq *} X_{ij}^2.$$

The mathematical program for matrix completion is given by

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times m}} \ & \frac{1}{2}\|X - M\|_{OB}^2 \\ \text{s.t.} \quad & \operatorname{rank}(X) \leq k. \end{aligned}$$

Since the constraint over the rank of a matrix is non-convex, it is standard to consider a relaxation that replaces the rank constraint by the nuclear norm. It is known that the nuclear norm is a lower bound on the matrix

rank if the singular values are bounded by one (see exercises). Thus, we arrive at the following convex program for matrix completion:

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times m}} \ & \frac{1}{2}\|X - M\|_{OB}^2 \\ \text{s.t.} \quad & \|X\|_* \leq k. \end{aligned} \tag{9.1}$$

We consider algorithms to solve this convex optimization problem next.

9.3 The Frank-Wolfe method

In this section we consider minimization of a convex function over a convex domain.

The conditional gradient (CG) method, or Frank-Wolfe algorithm, is a simple algorithm for minimizing a smooth convex function f over a convex set $\mathcal{K} \subseteq \mathbb{R}^n$. The appeal of the method is that it is a first order interior point method - the iterates always lie inside the convex set, and thus no projections are needed, and the update step on each iteration simply requires minimizing a linear objective over the set. The basic method is given in Algorithm 13.

Algorithm 13 Conditional gradient

1: Input: step sizes $\{\eta_t \in (0,1],\ t \in [T]\}$, initial point $\mathbf{x}_1 \in \mathcal{K}$.
2: **for** $t = 1$ to T **do**
3: $\quad \mathbf{v}_t \leftarrow \arg\min_{\mathbf{x} \in \mathcal{K}} \left\{\mathbf{x}^\top \nabla f(\mathbf{x}_t)\right\}$.
4: $\quad \mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \eta_t(\mathbf{v}_t - \mathbf{x}_t)$.
5: **end for**

Note that in the CG method, the update to the iterate $\mathbf{x}_t$ may be not be in the direction of the gradient, as $\mathbf{v}_t$ is the result of a linear optimization procedure in the direction of the negative gradient. This is depicted in Figure 9.1.

The following theorem gives an essentially tight performance guarantee of this algorithm over smooth functions. Recall our notation from Chapter 2: $\mathbf{x}^\star$ denotes the global minimizer of f over $\mathcal{K}$, D denotes the diameter of the set $\mathcal{K}$, and $h_t = f(\mathbf{x}_t) - f(\mathbf{x}^\star)$ denotes the suboptimality of the objective value in iteration t.

Theorem 9.1. *The CG algorithm applied to β-smooth functions with step sizes $\eta_t = \min\{\frac{2H}{t}, 1\}$, for $H \geq \max\{1, h_1\}$, attains the following convergence guarantee:*

$$h_t \leq \frac{2\beta H D^2}{t}$$

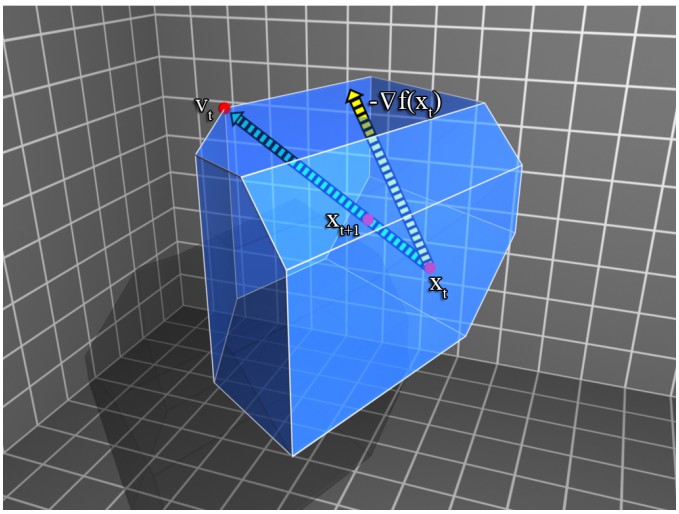

Figure 9.1: Direction of progression of the conditional gradient algorithm.

Proof. As done before in this manuscript, we denote $\nabla_t = \nabla f(\mathbf{x}_t)$, and also denote $H \geq \max\{h_1, 1\}$, such that $\eta_t = \min\{\frac{2H}{t}, 1\}$. For any set of step sizes, we have

$$
\begin{aligned}
& f(\mathbf{x}_{t+1}) - f(\mathbf{x}^\star) = f(\mathbf{x}_t + \eta_t(\mathbf{v}_t - \mathbf{x}_t)) - f(\mathbf{x}^\star) \\
& \leq f(\mathbf{x}_t) - f(\mathbf{x}^\star) + \eta_t(\mathbf{v}_t - \mathbf{x}_t)^\top \nabla_t + \eta_t^2 \frac{\beta}{2} \|\mathbf{v}_t - \mathbf{x}_t\|^2 \qquad \beta\text{-smoothness} \\
& \leq f(\mathbf{x}_t) - f(\mathbf{x}^\star) + \eta_t(\mathbf{x}^\star - \mathbf{x}_t)^\top \nabla_t + \eta_t^2 \frac{\beta}{2} \|\mathbf{v}_t - \mathbf{x}_t\|^2 \qquad \mathbf{v}_t \text{ optimality} \\
& \leq f(\mathbf{x}_t) - f(\mathbf{x}^\star) + \eta_t(f(\mathbf{x}^\star) - f(\mathbf{x}_t)) + \eta_t^2 \frac{\beta}{2} \|\mathbf{v}_t - \mathbf{x}_t\|^2 \quad \text{convexity of } f \\
& \leq (1 - \eta_t)(f(\mathbf{x}_t) - f(\mathbf{x}^\star)) + \frac{\eta_t^2 \beta}{2} D^2. \qquad (9.2)
\end{aligned}
$$

We reached the recursion $h_{t+1} \leq (1-\eta_t)h_t + \eta_t^2 \frac{\beta D^2}{2}$, and by induction,

$$\begin{aligned}
h_{t+1} &\leq (1-\eta_t)h_t + \eta_t^2 \frac{\beta D^2}{2} \\
&\leq (1-\eta_t)\frac{2\beta H D^2}{t} + \eta_t^2 \frac{\beta D^2}{2} && \text{induction hypothesis} \\
&\leq (1-\frac{2H}{t})\frac{2\beta H D^2}{t} + \frac{4H^2}{t^2}\frac{\beta D^2}{2} && \text{value of } \eta_t \\
&= \frac{2\beta H D^2}{t} - \frac{2H^2 \beta D^2}{t^2} \\
&\leq \frac{2\beta H D^2}{t}(1-\frac{1}{t}) && \text{since } H \geq 1 \\
&\leq \frac{2\beta H D^2}{t+1}. && \frac{t-1}{t} \leq \frac{t}{t+1}
\end{aligned}$$

□

9.4 Projections vs. linear optimization

The conditional gradient (Frank-Wolfe) algorithm described before does not resort to projections, but rather computes a linear optimization problem of the form

$$\arg\min_{\mathbf{x}\in\mathcal{K}} \left\{ \mathbf{x}^\top \mathbf{u} \right\}. \tag{9.3}$$

When is the CG method computationally preferable? The overall computational complexity of an iterative optimization algorithm is the product of the number of iterations and the computational cost per iteration. The CG method does not converge as well as the most efficient gradient descent algorithms, meaning it requires more iterations to produce a solution of a comparable level of accuracy. However, for many interesting scenarios the computational cost of a linear optimization step (9.3) is *significantly* lower than that of a projection step.

Let us point out several examples of problems for which we have very efficient linear optimization algorithms, whereas our state-of-the-art algorithms for computing projections are significantly slower.

Recommendation systems and matrix prediction. In the example pointed out in the preceding section of matrix completion, known methods

for projection onto the spectahedron, or more generally the bounded nuclear-norm ball, require singular value decompositions, which take superlinear time via our best known methods. In contrast, the CG method requires maximal eigenvector computations which can be carried out in linear time via the power method (or the more sophisticated Lanczos algorithm).

Network routing and convex graph problems. Various routing and graph problems can be modeled as convex optimization problems over a convex set called the flow polytope.

Consider a directed acyclic graph with m edges, a source node marked s and a target node marked t. Every path from s to t in the graph can be represented by its identifying vector, that is a vector in $\{0,1\}^m$ in which the entries that are set to 1 correspond to edges of the path. The flow polytope of the graph is the convex hull of all such identifying vectors of the simple paths from s to t. This polytope is also exactly the set of all unit s–t flows in the graph if we assume that each edge has a unit flow capacity (a flow is represented here as a vector in $\mathbb{R}^m$ in which each entry is the amount of flow through the corresponding edge).

Since the flow polytope is just the convex hull of s–t paths in the graph, minimizing a linear objective over it amounts to finding a minimum weight path given weights for the edges. For the shortest path problem we have very efficient combinatorial optimization algorithms, namely Dijkstra's algorithm.

Thus, applying the CG algorithm to solve **any** convex optimization problem over the flow polytope will only require iterative shortest path computations.

Ranking and permutations. A common way to represent a permutation or ordering is by a permutation matrix. Such are square matrices over $\{0,1\}^{n\times n}$ that contain exactly one 1 entry in each row and column.

Doubly-stochastic matrices are square, real-valued matrices with non-negative entries, in which the sum of entries of each row and each column amounts to 1. The polytope that defines all doubly-stochastic matrices is called the Birkhoff-von Neumann polytope. The Birkhoff-von Neumann theorem states that this polytope is the convex hull of exactly all $n \times n$ permutation matrices.

Since a permutation matrix corresponds to a perfect matching in a fully connected bipartite graph, linear minimization over this polytope corresponds to finding a minimum weight perfect matching in a bipartite graph.

Consider a convex optimization problem over the Birkhoff-von Neumann polytope. The CG algorithm will iteratively solve a linear optimization problem over the BVN polytope, thus iteratively solving a minimum weight perfect matching in a bipartite graph problem, which is a well-studied combinatorial optimization problem for which we know of efficient algorithms. In contrast, other gradient based methods will require projections, which are quadratic optimization problems over the BVN polytope.

Matroid polytopes. A matroid is pair (E, I) where E is a set of elements and I is a set of subsets of E called the independent sets which satisfy various interesting proprieties that resemble the concept of linear independence in vector spaces. Matroids have been studied extensively in combinatorial optimization and a key example of a matroid is the graphical matroid in which the set E is the set of edges of a given graph and the set I is the set of all subsets of E which are cycle-free. In this case, I contains all the spanning trees of the graph. A subset $S \in I$ could be represented by its identifying vector which lies in $\{0,1\}^{|E|}$ which also gives rise to the matroid polytope which is just the convex hull of all identifying vectors of sets in I. It can be shown that some matroid polytopes are defined by exponentially many linear inequalities (exponential in $|E|$), which makes optimization over them difficult.

On the other hand, linear optimization over matroid polytopes is easy using a simple greedy procedure which runs in nearly linear time. Thus, the CG method serves as an efficient algorithm to solve any convex optimization problem over matroids iteratively using only a simple greedy procedure.

9.5 Exercises

1. Prove that if the singular values are smaller than or equal to one, then the nuclear norm is a lower bound on the rank, i.e., show

$$\text{rank}(X) \geq \|X\|_*.$$

2. Prove that the trace is related to the nuclear norm via

$$\|X\|_* = \mathbf{Tr}(\sqrt{XX^\top}) = \mathbf{Tr}(\sqrt{X^\top X}).$$

3. Show that maximizing a linear function over the spectahedron is equivalent to a maximal eigenvector computation. That is, show that the following mathematical program:

$$\begin{aligned} &\min X \bullet C \\ &X \in S_d = \{X \in \mathbb{R}^{d \times d} \ , \ X \succcurlyeq 0 \ , \ \mathbf{Tr}(X) \leq 1\}, \end{aligned}$$

   is equivalent to the following:

$$\begin{aligned} &\min_{\mathbf{x} \in \mathbb{R}^d} \mathbf{x}^\top C \mathbf{x} \\ &\text{s.t. } \|\mathbf{x}\|_2 \leq 1. \end{aligned}$$

4. Download the MovieLens dataset from the web. Implement an online recommendation system based on the matrix completion model: implement the OCG and OGD algorithms for matrix completion. Benchmark your results.

9.6 Bibliographic Remarks

The matrix completion model has been extremely popular since its inception in the context of recommendation systems [80, 66, 69, 50, 14, 75].

The conditional gradient algorithm was devised in the seminal paper by Frank and Wolfe [21]. Due to the applicability of the FW algorithm to large-scale constrained problems, it has been a method of choice in recent machine learning applications, to name a few: [42, 49, 41, 20, 30, 36, 72, 7, 82, 22, 23, 8].

The online conditional gradient algorithm is due to [36]. An optimal regret algorithm, attaining the $O(\sqrt{T})$ bound, for the special case of polyhedral sets was devised in [23].

Chapter 10

Second order methods for machine learning

At this point in our course, we have exhausted the main techniques in first-order (or gradient-based) optimization. We have studied the main workhorse - stochastic gradient descent, the three acceleration techniques, and projection-free gradient methods. Have we exhausted optimization for ML?

In this section we discuss using higher derivatives of the objective function to accelerate optimization. The canonical method is Newton's method, which involves the second derivative or Hessian in high dimensions. The vanilla approach is computationally expensive since it involves matrix inversion in high dimensions that machine learning problems usually require.

However, recent progress in random estimators gives rise to linear-time second order methods, for which each iteration is as computationally cheap as gradient descent.

10.1 Motivating example: linear regression

In the problem of linear regression we are given a set of measurements $\{\mathbf{a}_i \in \mathbb{R}^d, b_i \in \mathbb{R}\}$, and the goal is to find a set of weights that explains them best in the mean squared error sense. As a mathematical program, the goal is to optimize:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \left\{ \frac{1}{2} \sum_{i\in[m]} \left(\mathbf{a}_i^\top \mathbf{x} - b_i \right)^2 \right\},$$

or in matrix form,

$$\min_{\mathbf{x}} f(\mathbf{x}) = \left\{ \frac{1}{2} \|A\mathbf{x} - \mathbf{b}\|^2 \right\}.$$

Here $A \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$. Notice that the objective function f is smooth, but not necessarily strongly convex. Therefore, all algorithms that we have studied so far without exception, which are all first order methods, attain rates which are poly$(\frac{1}{\varepsilon})$.

However, the linear regression problem has a closed form solution that can be computed by taking the gradient to be zero, i.e. $(A\mathbf{x} - \mathbf{b})^\top A = 0$, which gives

$$\mathbf{x} = (A^\top A)^{-1} A^\top \mathbf{b}.$$

The Newton direction is given by the inverse Hessian multiplied by the gradient, $\nabla^{-2} f(\mathbf{x}) \nabla f(\mathbf{x})$. Observe that a single Newton step, i.e. moving in the Newton direction with step size one, from any direction gets us directly to the optimal solution in one iteration! (see exercises)

More generally, Newton's method yields $O(\log \frac{1}{\varepsilon})$ convergence rates for a large class of functions without dependence on the condition number of the function! We study this property next.

10.2 Self-Concordant Functions

In this section we define and collect some of the properties of a special class of functions, called self-concordant functions. These functions allow Newton's method to run in time which is independent of the condition number. The class of self-concordant functions is expressive and includes quadratic functions, logarithms of inner products, a variety of barriers such as the log determinant, and many more.

An excellent reference for this material is the lecture notes on this subject by Nemirovski [55]. We begin by defining self-concordant functions.

Definition 10.1 (Self-Concordant Functions)**.** *Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a non-empty open convex set, and and let $f : \mathcal{K} \mapsto \mathbb{R}$ be a C^3 convex function. Then, f is said to be self-concordant if*

$$|\nabla^3 f(\mathbf{x})[\mathbf{h}, \mathbf{h}, \mathbf{h}]| \leq 2(\mathbf{h}^\top \nabla^2 f(\mathbf{x}) \mathbf{h})^{3/2},$$

where we have

$$\nabla^k f(\mathbf{x})[\mathbf{h}_1, \ldots, \mathbf{h}_k] \triangleq \frac{\partial^k}{\partial t_1 \ldots \partial t_k} |_{t_1 = \cdots = t_k} f(\mathbf{x} + t_1 \mathbf{h}_1 + \cdots + t_k \mathbf{h}_k).$$

Another key object in the analysis of self concordant functions is the notion of a Dikin Ellipsoid, which is the unit ball around a point in the norm given by the Hessian $\|\cdot\|_{\nabla^2 f}$ at the point. We will refer to this norm as the *local norm* around a point and denote it as $\|\cdot\|_{\mathbf{x}}$. Formally,

Definition 10.2 (Dikin ellipsoid)**.** *The Dikin ellipsoid of radius r centered at a point $\mathbf{x}$ is defined as*

$$\mathcal{E}_r(\mathbf{x}) \triangleq \{\mathbf{y} \mid \|\mathbf{y}-\mathbf{x}\|_{\nabla^2 f(\mathbf{x})} \leq r\}$$

One of the key properties of self-concordant functions that we use is that inside the Dikin ellipsoid, the function is well conditioned with respect to the local norm at the center. The next lemma makes this formal. The proof of this lemma can be found in [55].

Lemma 10.3 (See [55])**.** *For all $\mathbf{h}$ such that $\|\mathbf{h}\|_{\mathbf{x}} < 1$ we have that*

$$(1-\|\mathbf{h}\|_{\mathbf{x}})^2 \nabla^2 f(\mathbf{x}) \preceq \nabla^2 f(\mathbf{x}+\mathbf{h}) \preceq \frac{1}{(1-\|\mathbf{h}\|_{\mathbf{x}})^2}\nabla^2 f(\mathbf{x})$$

Another key quantity, which is used both as a potential function as well as a dampening for the step size in the analysis of Newton's method, is the Newton Decrement:

$$\lambda_{\mathbf{x}} \triangleq \|\nabla f(\mathbf{x})\|_{\mathbf{x}}^* = \sqrt{\nabla f(\mathbf{x})^\top \nabla^{-2} f(\mathbf{x}) \nabla f(\mathbf{x})}.$$

The following lemma quantifies how $\lambda_{\mathbf{x}}$ behaves as a potential by showing that once it drops below 1, it ensures that the minimum of the function lies in the current Dikin ellipsoid. This is the property which we use crucially in our analysis. The proof can be found in [55].

Lemma 10.4 (See [55])**.** *If $\lambda_{\mathbf{x}} < 1$ then*

$$\|\mathbf{x}-\mathbf{x}^*\|_{\mathbf{x}} \leq \frac{\lambda_{\mathbf{x}}}{1-\lambda_{\mathbf{x}}}$$

10.3 Newton's method for self-concordant functions

Before introducing the linear time second order methods, we start by introducing a robust Newton's method and its properties. The pseudo-code is given in Algorithm 14.

The usual analysis of Newton's method allows for quadratic convergence, i.e. error ε in $O(\log\log\frac{1}{\varepsilon})$ iterations for convex objectives. However, we prefer to present a version of Newton's method which is robust to certain random estimators of the Newton direction. This yields a slower rate of $O(\log\frac{1}{\varepsilon})$. The faster running time per iteration, which does not require matrix manipulations, more than makes up for this.

Algorithm 14 Robust Newton's method

Input: $T, \mathbf{x}_1$
for $t = 1$ to T **do**
  Set $c = \frac{1}{8}$, $\eta = \min\{c, \frac{c}{8\lambda_{\mathbf{x}_t}}\}$. Let $\frac{1}{2}\nabla^{-2}f(\mathbf{x}_t) \preceq \tilde{\nabla}_t^{-2} \preceq 2\nabla^{-2}f(\mathbf{x}_t)$.
  $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\tilde{\nabla}_t^{-2}\nabla f(\mathbf{x}_t)$
end for
return $\mathbf{x}_{T+1}$

It is important to notice that every two consecutive points are within the same Dikin ellipsoid of radius $\frac{1}{2}$. Denote $\nabla_t = \nabla_{\mathbf{x}_t}$, and similarly for the Hessian. Then we have:

$$\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_{\mathbf{x}_t}^2 = \eta^2\nabla_t^\top\tilde{\nabla}_t^{-2}\nabla_t^2\tilde{\nabla}_t^{-2}\nabla_t \le 4\eta^2\lambda_t^2 \le \frac{1}{2}.$$

The advantage of Newton's method as applied to self-concordant functions is its linear convergence rate, as given in the following theorem.

Theorem 10.5. *Let f be self-concordant, and $f(\mathbf{x}_1) \le M$, then*

$$h_t = f(\mathbf{x}_t) - f(\mathbf{x}^*) \le O(M + \log\frac{1}{\varepsilon})$$

The proof of this theorem is composed of two steps, according to the magnitude of the Newton decrement.

Phase 1: damped Newton

Lemma 10.6. *As long as $\lambda_{\mathbf{x}} \ge \frac{1}{8}$, we have that*

$$h_t \le -\frac{1}{4}c$$

Proof. Using similar analysis to the descent lemma we have that

$$\begin{aligned}
& f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) \\
&\leq \nabla_t^\top (\mathbf{x}_{t+1} - \mathbf{x}_t) + \tfrac{1}{2}(\mathbf{x}_t - \mathbf{x}_{t+1})^\top \nabla^2(\zeta)(\mathbf{x}_t - \mathbf{x}_{t+1}) && \text{Taylor} \\
&\leq \nabla_t^\top (\mathbf{x}_{t+1} - \mathbf{x}_t) + \tfrac{1}{4}(\mathbf{x}_t - \mathbf{x}_{t+1})^\top \nabla^2(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}_{t+1}) && \mathbf{x}_{t+1} \in \mathcal{E}_{1/2}(\mathbf{x}_t) \\
&= -\eta \nabla_t^\top \tilde{\nabla}_t^{-2} \nabla_t + \tfrac{1}{4}\eta^2 \nabla_t^\top \tilde{\nabla}_t^{-2} \nabla_t^2 \tilde{\nabla}_t^{-2} \nabla_t \\
&= -\eta \lambda_t^2 + \tfrac{1}{4}\eta^2 \lambda_t^2 \leq -\tfrac{1}{16} c
\end{aligned}$$

□

The conclusion from this step is that after $O(M)$ steps, Algorithm 14 reaches a point for which $\lambda_{\mathbf{x}} \leq \frac{1}{8}$. According to Lemma 10.4, we also have that $\|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{x}} \leq \frac{1}{4}$, that is, the optimum is in the same Dikin ellipsoid as the current point.

Phase 2: pure Newton In the second phase our step size is changed to be larger. In this case, we are guaranteed that the Newton decrement is less than one, and thus we know that the global optimum is in the same Dikin ellipsoid as the current point. In this ellipsoid, all Hessians are equivalent up to a factor of two, and thus Mirrored-Descent with the inverse Hessian as preconditioner becomes gradient descent. We make this formal below.

Algorithm 15 Preconditioned Gradient Descent

Input: P, T
for $t = 1$ to T **do**
  $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta P^{-1} \nabla f(\mathbf{x}_t)$
end for
return $\mathbf{x}_{T+1}$

Lemma 10.7. *Suppose that* $\frac{1}{2}P \preceq \nabla^2 f(\mathbf{x}) \preceq 2P$, *and* $\|\mathbf{x}_1 - \mathbf{x}^*\|_P \leq \frac{1}{2}$, *then Algorithm 15 converges as*

$$h_{t+1} \leq h_1 e^{-\frac{1}{8}t}.$$

This theorem follows from noticing that the function $g(\mathbf{z}) = f(P^{-1/2}\mathbf{x})$ is $\frac{1}{2}$-strongly convex and 2-smooth, and using Theorem 3.2. It can be shown that gradient descent on g is equivalent to Newton's method in f. Details are left as an exercise.

An immediate corollary is that Newton's method converges at a rate of $O(\log \frac{1}{\varepsilon})$ in this phase.

10.4 Linear-time second-order methods

Newton's algorithm is of foundational importance in the study of mathematical programming in general. A major application are interior point methods for convex optimization, which are the most important polynomial-time algorithms for general constrained convex optimization.

However, the main downside of this method is the need to maintain and manipulate matrices - namely the Hessians. This is completely impractical for machine learning applications in which the dimension is huge.

Another significant downside is the non-robust nature of the algorithm, which makes applying it in stochastic environments challenging.

In this section we show how to apply Newton's method to machine learning problems. This involves relatively new developments that allow for linear-time per-iteration complexity, similar to SGD, and theoretically superior running times. At the time of writing, however, these methods are practical only for convex optimization, and have not shown superior performance on optimization tasks involving deep neural networks.

The first step to developing a linear time Newton's method is an efficient stochastic estimator for the Newton direction, and the Hessian **inverse**.

10.4.1 Estimators for the Hessian Inverse

The key idea underlying the construction is the following well known fact about the Taylor series expansion of the matrix inverse.

Lemma 10.8. *For a matrix $A \in \mathbb{R}^{d\times d}$ such that $A \succeq 0$ and $\|A\| \leq 1$, we have that*

$$A^{-1} = \sum_{i=0}^{\infty} (I - A)^i$$

We propose two unbiased estimators based on the above series. To define the first estimator pick a probability distribution over non-negative integers $\{p_i\}$ and sample $\hat{i}$ from the above distribution. Let $X_1, \ldots X_{\hat{i}}$ be independent samples of the Hessian $\nabla^2 f$ and define the estimator as

Definition 10.9 (Estimator 1)**.**

$$\tilde{\nabla}^{-2} f = \frac{1}{p_{\hat{i}}} \prod_{j=1}^{\hat{i}} (I - X_j)$$

Observe that our estimator of the Hessian inverse is unbiased, i.e. $\mathbf{E}[\hat{X}] = \nabla^{-2} f$ at any point. Estimator 1 has the disadvantage that in a single sample it incorporates only one term of the Taylor series.

The second estimator below is based on the observation that the above series has the following succinct recursive definition, and is more efficient.

For a matrix A define

$$A_j^{-1} = \sum_{i=0}^{j} (I - A)^i$$

i.e. the first j terms of the above Taylor expansion. It is easy to see that the following recursion holds for A_j^{-1}

$$A_j^{-1} = I + (I - A)A_{j-1}^{-1}$$

Using the above recursive formulation, we now describe an unbiased estimator of $\nabla^{-2} f$ by deriving an unbiased estimator $\tilde{\nabla}^{-2} f_j$ for $\nabla^{-2} f_j$.

Definition 10.10 (Estimator 2)**.** *Given j independent and unbiased samples $\{X_1 \dots X_j\}$ of the hessian $\nabla^2 f$. Define $\{\tilde{\nabla}^{-2} f_0 \dots \tilde{\nabla}^{-2} f_j\}$ recursively as follows*

$$\tilde{\nabla}^{-2} f_0 = I$$

$$\tilde{\nabla}^{-2} f_t = I + (I - X_j)\tilde{\nabla}^{-2} f_{t-1}$$

It can be readily seen that $\mathbf{E}[\tilde{\nabla}^{-2} f_j] = \nabla^{-2} f_j$ and therefore $\mathbf{E}[\tilde{\nabla}^{-2} f_j] \to \nabla^{-2} f$ as $j \to \infty$ giving us an unbiased estimator in the limit.

10.4.2 Incorporating the estimator

Both of the above estimators can be computed using only Hessian-vector products, rather than matrix manipulations. For many machine learning problems, Hessian-vector products can be computed in linear time. Examples include:

1. Convex regression and SVM objectives over training data have the form

   $$\min_{\mathbf{w}} f(\mathbf{w}) = \mathop{\mathbf{E}}_{i}[\ell(\mathbf{w}^\top \mathbf{x}_i)],$$

   where ℓ is a convex function. The Hessian can thus be written as

   $$\nabla^2 f(\mathbf{w}) = \mathop{\mathbf{E}}_{i}[\ell''(\mathbf{w}^\top \mathbf{x}_i)\mathbf{x}_i \mathbf{x}_i^\top]$$

Thus, the first Newton direction estimator can now be written as

$$\tilde{\nabla}^2 f(\mathbf{w}) \nabla_{\mathbf{w}} = \mathop{\mathbf{E}}_{j \sim \mathcal{D}} [\prod_{i=1}^{j} (I - \ell''(\mathbf{w}^\top \mathbf{x}_i) \mathbf{x}_i \mathbf{x}_i^\top)] \nabla_{\mathbf{w}}.$$

Notice that this estimator can be computed using j vector-vector products if the ordinal j was randomly chosen.

2. Non-convex optimization over neural networks: a similar derivation as above shows that the estimator can be computed only using Hessian-vector products. The special structure of neural networks allow this computation in a constant number of backpropagation steps, i.e. linear time in the network size, this is called the "Pearlmutter trick", see [61].

   We note that non-convex optimization presents special challenges for second order methods, since the Hessian need not be positive semi-definite. Nevertheless, the techniques presented hereby can still be used to provide theoretical speedups for second order methods over first order methods in terms of convergence to local minima. The details are beyond our scope, and can be found in [2].

Putting everything together. These estimators we have studied can be used to create unbiased estimators to the Newton direction of the form $\tilde{\nabla}_{\mathbf{x}}^{-2} \nabla_x$ for $\tilde{\nabla}_{\mathbf{x}}^{-2}$ which satisfies

$$\frac{1}{2} \nabla^{-2} f(\mathbf{x}_t) \preceq \tilde{\nabla}_t^{-2} \preceq 2 \nabla^{-2} f(\mathbf{x}_t).$$

These can be incorporated into Algorithm 14, which we proved is capable of obtaining fast convergence with approximate Newton directions of this form.

10.5 Exercises

1. Prove that a single Newton step for linear regression yields the optimal solution.

2. Let $f : \mathbb{R}^d \mapsto \mathbb{R}$, and consider the affine transformation $\mathbf{y} = A\mathbf{x}$, for $A \in \mathbb{R}^{d \times d}$ being a symmetric matrix. Prove that

$$\mathbf{y}_{t+1} \leftarrow \mathbf{y}_t - \eta \nabla f(\mathbf{y}_t)$$

   is equivalent to

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta A^{-2} \nabla f(\mathbf{x}_t).$$

3. Prove that the function $g(\mathbf{z})$ defined in phase 2 of the robust Newton algorithm is $\frac{1}{2}$-strongly convex and 2-smooth. Conclude with a proof of Theorem 10.7.

10.6 Bibliographic Remarks

The modern application of Newton's method to convex optimization was put forth in the seminal work of Nesterov and Nemirovski [58] on interior point methods. A wonderful exposition is Nemirovski's lecture notes [55].

The fact that Hessian-vector products can be computed in linear time for feed forward neural networks was described in [61]. Linear time second order methods for machine learning and the Hessian-vector product model in machine learning was introduced in [4]. This was extended to non-convex optimization for deep learning in [2].

Chapter 11

Hyperparameter Optimization

Thus far in this class, we have been talking about continuous mathematical optimization, where the search space of our optimization problem is continuous and mostly convex. For example, we have learned about how to optimize the weights of a deep neural network, which take continuous real values, via various optimization algorithms (SGD, AdaGrad, Newton's method, etc.).

However, in the process of training a neural network, there are some meta parameters, which we call *hyperparameters*, that have a profound effect on the final outcome. These are global, mostly discrete, parameters that are treated differently by algorithm designers as well as by engineers. Examples include the architecture of the neural network (number of layers, width of each layer, type of activation function, ...), the optimization scheme for updating weights (SGD/AdaGrad, initial learning rate, decay rate of learning rate, momentum parameter, ...), and many more. Roughly speaking, these hyperparameters are chosen before the training starts.

The purpose of this chapter is to formalize this problem as an optimization problem in machine learning, which requires a different methodology than we have treated in the rest of this course. We remark that hyperparameter optimization is still an active area of research and its theoretical properties are not well understood as of this time.

11.1 Formalizing the problem

What makes hyperparameters different from "regular" parameters?

1. The search space is often discrete (for example, number of layers). As such, there is no natural notion of gradient or differentials and it is not clear how to apply the iterative methods we have studied thus far.
2. Even evaluating the objective function is extremely expensive (think of evaluating the test error of the trained neural network). Thus it is crucial to minimize the number of function evaluations, whereas other computations are significantly less expensive.
3. Evaluating the function can be done in parallel. As an example, training feedforward deep neural networks over different architectures can be done in parallel.

More formally, we consider the following optimization problem

$$\min_{\mathbf{x}_i \in GF(q_i)} \quad f(\mathbf{x}),$$

where $\mathbf{x}$ is the representation of discrete hyperparameters, each taking value from $q_i \geq 2$ possible discrete values and thus in $GF(q)$, the Galois field of order q. The example to keep in mind is that the objective $f(\mathbf{x})$ is the test error of the neural network trained with hyperparameters $\mathbf{x}$. Note that $\mathbf{x}$ has a search space of size $\prod_i q_i \geq 2^n$, exponentially large in the number of different hyperparameters.

11.2 Hyperparameter optimization algorithms

The properties of the problem mentioned before prohibits the use of the algorithms we have studied thus far, which are all suitable for continuous optimization. A naive method is to perform a grid search over all hyperparameters, but this quickly becomes infeasible. An emerging field of research in recent years, called *AutoML*, aims to choose hyperparameters automatically. The following techniques are in common use:

- **Grid search**, try all possible assignments of hyperparameters and return the best. This becomes infeasible very quickly with n - the number of hyperparameters.
- **Random search**, where one randomly picks some choices of hyperparameters, evaluates their function objective, and chooses the one choice of hyperparameters giving best performance. An advantage of this method is that it is easy to implement in parallel.

- **Successive Halving and Hyperband**, random search combined with early stopping using multi-armed bandit techniques. These gain a small constant factor improvement over random search.
- **Bayesian optimization**, a statistical approach which has a prior over the objective and tries to iteratively pick an evaluation point which reduces the variance in objective value. Finally it picks the point that attains the lowest objective objective with highest confidence. This approach is sequential in nature and thus difficult to parallelize. Another important question is how to choose a good prior.

The hyperparameter optimization problem is essentially a combinatorial optimization problem with exponentially large search space. Without further assumptions, this optimization problem is information-theoretically hard. Such assumptions are explored in the next section with an accompanying algorithm.

Finally, we note that a simple but hard-to-beat benchmark is random search with double budget. That is, compare the performance of a method to that of random search, but allow random search double the query budget of your own method.

11.3 A Spectral Method

For simplicity, in this section we consider the case in which hyperparameters are binary. This retains the difficulty of the setting, but makes the mathematical derivation simpler. The optimization problem now becomes

$$\min_{\mathbf{x}\in\{-1,1\}^n} \quad f(\mathbf{x}). \tag{11.1}$$

The method we describe in this section is inspired by the following key observation: *although the whole search space of hyperparameters is exponentially large, it is often the case in practice that only a few hyperparameters together play a significant role in the performance of a deep neural network.*

To make this intuition more precise, we need some definitions and facts from Fourier analysis of Boolean functions.

Fact 11.1. *Any function $f : \{-1,1\}^n \to [-1,1]$ can be uniquely represented in the Fourier basis*

$$f(\mathbf{x}) = \sum_{S\subseteq[n]} \alpha_s \hat{\chi}_S(\mathbf{x}),$$

where each Fourier basis function

$$\hat{\chi}_S(\mathbf{x}) = \prod_{i \in S} x_i.$$

is a monomial, and thus $f(\mathbf{x})$ has a polynomial representation.

Now we are ready to formalize our key observation in the following assumption:

Assumption 11.2. *The objective function f in the hyperparameter optimization problem* (11.1) *is low degree and sparse in the Fourier basis, i.e.*

$$f(x) \approx \sum_{|S| \leq d} \alpha_S \hat{\chi}_S(x), \quad \|\boldsymbol{\alpha}\|_1 \leq k, \tag{11.2}$$

where d is the upper bound of polynomial degree, and k is the sparsity of Fourier coefficient $\boldsymbol{\alpha}$ (indexed by S) in ℓ_1 sense (which is a convex relaxation of $\|\boldsymbol{\alpha}\|_0$, the true sparsity).

Remark 11.3. *Clearly this assumption does not always hold. For example, many deep reinforcement learning algorithms nowadays rely heavily on the choice of the random seed, which can also be seen as a hyperparameter. If $\mathbf{x} \in \{-1, 1\}^{32}$ is the bit representation of a int32 random seed, then there is no reason to assume that a few of these bits should play a more significant role than the others.*

Under this assumption, all we need to do now is to find out the few important sets of variables S's, as well as their coefficients α_S's, in the approximation (11.2). Fortunately, there is already a whole area of research, called *compressed sensing*, that aims to recover a high-dimensional but sparse vector, using only a few linear measurements. Next, we will briefly introduce the problem of compressed sensing, and one useful result from the literature. After that, we will introduce the Harmonica algorithm, which applies compressed sensing techniques to solve the hyperparameter optimization problem (11.1).

11.3.1 Background: Compressed Sensing

The problem of compressed sensing is as follows. Suppose there is a hidden signal $\mathbf{x} \in \mathbb{R}^n$ that we cannot observe. In order to recover $\mathbf{x}$, we design a measurement matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, and obtain noisy linear measurements $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\eta} \in \mathbb{R}^m$, where $\boldsymbol{\eta}$ is some random noise. The difficulty arises when we

have a limited budget for measurements, i.e. $m \ll n$. Note that even without noise, recovering $\mathbf{x}$ is non-trivial since $\mathbf{y} = \mathbf{A}\mathbf{x}$ is an underdetermined linear system, therefore if there is one solution $\mathbf{x}$ that solves this linear system, there will be infinitely many solutions. The key to this problem is to assume that $\mathbf{x}$ is k-sparse, that is, $\|\mathbf{x}\|_0 \leq k$. This assumption has been justified in various real-world applications; for example, natural images tend to be sparse in the Fourier/wavelet domain, a property which forms the bases of many image compression algorithms.

Under the assumption of sparsity, the natural way to recover $\mathbf{x}$ is to solve a least squares problem, subject to some sparsity constraint $\|\mathbf{x}\|_0 \leq k$. However, ℓ_0 norm is difficult to handle, and it is often replaced by ℓ_1 norm, its convex relaxation. One useful result from the literature of compressed sensing is the following.

Proposition 11.4 (Informal statement of Theorem 4.4 in [63])**.** *Assume the ground-truth signal $\mathbf{x} \in \mathbb{R}^n$ is k-sparse. Then, with high probability, using a randomly designed $\mathbf{A} \in \mathbb{R}^{m \times n}$ that is "near-orthogonal" (random Gaussian matrix, subsampled Fourier basis, etc.), with $m = O(k \log(n)/\varepsilon)$ and $\|\boldsymbol{\eta}\|_2 = O(\sqrt{m})$, $\mathbf{x}$ can be recovered by a convex program*

$$\min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{y} - \mathbf{A}\mathbf{z}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{z}\|_1 \leq k, \tag{11.3}$$

with accuracy $\|\mathbf{x} - \mathbf{z}\|_2 \leq \varepsilon$.

This result is remarkable; in particular, it says that the number of measurements needed to recover a sparse signal is independent of the dimension n (up to a logarithm term), but only depends on the sparsity k and the desired accuracy ε. [1]

Remark 11.5. *The convex program* (11.3) *is equivalent to the following LASSO problem*

$$\min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{y} - \mathbf{A}\mathbf{z}\|_2^2 + \lambda \|\mathbf{z}\|_1,$$

with a proper choice of regularization parameter λ. The LASSO problem is an unconstrained convex program, and has efficient solvers, as per the algorithms we have studied in this course.

[1] It also depends on the desired high-probability bound, which is omitted in this informal statement.

11.3.2 The Spectral Algorithm

The main idea is that, under Assumption 11.2, we can view the problem of hyperparameter optimization as recovering the sparse signal $\boldsymbol{\alpha}$ from linear measurements. More specifically, we need to query T random samples, $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_T)$, and then solve the LASSO problem

$$\min_{\boldsymbol{\alpha}} \quad \sum_{t=1}^{T} \Big(\sum_{|S| \le d} \alpha_S \hat{\chi}_S(\mathbf{x}_t) - f(\mathbf{x}_t) \Big)^2 + \lambda \|\boldsymbol{\alpha}\|_1, \tag{11.4}$$

where the regularization term $\lambda\|\boldsymbol{\alpha}\|_1$ controls the sparsity of $\boldsymbol{\alpha}$. Also note that the constraint $|S| \le d$ not only implies that the solution is a low-degree polynomial, but also helps to reduce the "effective" dimension of $\boldsymbol{\alpha}$ from 2^n to $O(n^d)$, which makes it feasible to solve this LASSO problem.

Denote by $S_1, \ldots, S_s$ the indices of the s largest coefficients of the LASSO solution, and define

$$g(\mathbf{x}) = \sum_{i \in [s]} \alpha_{S_i} \hat{\chi}_{S_i}(\mathbf{x}),$$

which involves only a few dimensions of $\mathbf{x}$ since the LASSO solution is sparse and low-degree. The next step is to set the variables outside $\cup_{i \in [s]} S_i$ to arbitrary values, and compute a minimizer $\mathbf{x}^* \in \arg\min g(\mathbf{x})$. In other words, we have reduced the original problem of optimizing $f(\mathbf{x})$ over n variables, to the problem of optimizing $g(\mathbf{x})$ (an approximation of $f(\mathbf{x})$) over only a few variables (which is now feasible to solve). One remarkable feature of this algorithm is that the returned solution $\mathbf{x}^*$ may not belong to the samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, which is not the case for other existing methods (such as random search).

Using theoretical results from compressed sensing (e.g. Proposition 11.4), we can derive the following guarantee for the sparse recovery of $\boldsymbol{\alpha}$ via LASSO.

Theorem 11.6 (Informal statement of Lemma 7 in [38])**.** *Assume f is k-sparse in the Fourier expansion. Then, with $T = O(k^2 \log(n)/\varepsilon)$ samples, the solution of the LASSO problem* (11.4) *achieves ε accuracy.*

Finally, the above derivation can be considered as only one stage in a multi-stage process, each iteratively setting the value of a few more variables that are the most significant.

11.4 Bibliographic Remarks

For a nice exposition on hyperparameter optimization see [64, 65], in which the the benchmark of comparing to Random Search with double queries was proposed.

Perhaps the simplest approach to HPO is random sampling of different choices of parameters and picking the best amongst the chosen evaluations [9]. Successive Halving (SH) algorithm was introduced [43]. Hyperband further improves SH by automatically tuning the hyperparameters in SH [51].

The Bayesian optimization (BO) methodology is currently the most studied in HPO. For recent studies and algorithms of this flavor see [10, 78, 81, 79, 24, 84, 40].

The spectral approach for hyperparameter optimization was introduced in [38]. For an in-depth treatment of compressed sensing see the survey of [63], and for Fourier analysis of Boolean functions see [59].

Bibliography

[1] Jacob Abernethy, Elad Hazan, and Alexander Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. In *Proceedings of the 21st Annual Conference on Learning Theory*, pages 263–274, 2008.

[2] Naman Agarwal, Zeyuan Allen-Zhu, Brian Bullins, Elad Hazan, and Tengyu Ma. Finding approximate local minima faster than gradient descent. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1195–1199. ACM, 2017.

[3] Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. The case for full-matrix adaptive regularization. *arXiv preprint arXiv:1806.02958*, 2018.

[4] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

[5] Zeyuan Allen-Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. *arXiv preprint arXiv:1407.1537*, 2014.

[6] Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory-efficient adaptive optimization for large-scale learning. *arXiv preprint arXiv:1901.11150*, 2019.

[7] Francis Bach, Simon Lacoste-Julien, and Guillaume Obozinski. On the equivalence between herding and conditional gradient algorithms. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1359–1366, New York, NY, USA, July 2012. Omnipress.

[8] Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. Distributed frank-wolfe algorithm: A unified framework for communication-efficient sparse learning. *CoRR*, abs/1404.2644, 2014.

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[10] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[11] J.M. Borwein and A.S. Lewis. *Convex Analysis and Nonlinear Optimization: Theory and Examples*. CMS Books in Mathematics. Springer, 2006.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[13] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3–4):231–357, 2015.

[14] E. Candes and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9:717–772, 2009.

[15] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[16] Xinyi Chen, Naman Agarwal, Elad Hazan, Cyril Zhang, and Yi Zhang. Extreme tensoring for low-memory preconditioning. *arXiv preprint arXiv:1902.04620*, 2019.

[17] Qi Deng, Yi Cheng, and Guanghui Lan. Optimal adaptive and accelerated stochastic gradient descent. *arXiv preprint arXiv:1810.00553*, 2018.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[19] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, pages 257–269, 2010.

[20] Miroslav Dudík, Zaïd Harchaoui, and Jérôme Malick. Lifted coordinate descent for learning with trace-norm regularization. *Journal of Machine Learning Research - Proceedings Track*, 22:327–336, 2012.

[21] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:149–154, 1956.

[22] Dan Garber and Elad Hazan. Approximating semidefinite programs in sublinear time. In *NIPS*, pages 1080–1088, 2011.

[23] Dan Garber and Elad Hazan. Playing non-linear games with linear oracles. In *FOCS*, pages 420–428, 2013.

[24] Jacob R. Gardner, Matt J. Kusner, Zhixiang Eddie Xu, Kilian Q. Weinberger, and John P. Cunningham. Bayesian optimization with inequality constraints. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 937–945, 2014.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] A .J. Grove, N. Littlestone, and D. Schuurmans. General convergence results for linear discriminant updates. *Machine Learning*, 43(3):173–210, 2001.

[27] Vineet Gupta, Tomer Koren, and Yoram Singer. A unified approach to adaptive regularization in online and stochastic optimization. *arXiv preprint arXiv:1706.06569*, 2017.

[28] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. *arXiv preprint arXiv:1802.09568*, 2018.

[29] James Hannan. Approximation to bayes risk in repeated play. *In M. Dresher, A. W. Tucker, and P. Wolfe, editors, Contributions to the Theory of Games, volume 3*, pages 97–139, 1957.

[30] Zaïd Harchaoui, Matthijs Douze, Mattis Paulin, Miroslav Dudík, and Jérôme Malick. Large-scale image classification with trace-norm regularization. In *CVPR*, pages 3386–3393, 2012.

[31] Elad Hazan. Introduction to online convex optimization. *Foundations and TrendsÂ® in Optimization*, 2(3-4):157–325, 2016.

[32] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. In *Machine Learning*, volume 69(2–3), pages 169–192, 2007.

[33] Elad Hazan and Sham Kakade. Revisiting the polyak step size. *arXiv preprint arXiv:1905.00313*, 2019.

[34] Elad Hazan and Satyen Kale. Extracting certainty from uncertainty: Regret bounded by variation in costs. In *The 21st Annual Conference on Learning Theory (COLT)*, pages 57–68, 2008.

[35] Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. *Journal of Machine Learning Research - Proceedings Track*, pages 421–436, 2011.

[36] Elad Hazan and Satyen Kale. Projection-free online learning. In *ICML*, 2012.

[37] Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *The Journal of Machine Learning Research*, 15(1):2489–2512, 2014.

[38] Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *ICLR*, 2018.

[39] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14, 2012.

[40] Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine Annette Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic RBF surrogates. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 822–829, 2017.

[41] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *ICML*, 2013.

[42] Martin Jaggi and Marek Sulovský. A simple algorithm for nuclear norm regularized problems. In *ICML*, pages 471–478, 2010.

[43] Kevin G. Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 240–248, 2016.

[44] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[45] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[46] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Inf. Comput.*, 132(1):1–63, 1997.

[48] Jyrki Kivinen and Manfred K. Warmuth. Relative loss bounds for multidimensional regression problems. *Machine Learning*, 45(3):301–329, 2001.

[49] Simon Lacoste-Julien, Martin Jaggi, Mark W. Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 53–61, 2013.

[50] J. Lee, B. Recht, R. Salakhutdinov, N. Srebro, and J. A. Tropp. Practical large-scale optimization for max-norm regularization. In *NIPS*, pages 1297–1305, 2010.

[51] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *ArXiv e-prints*, March 2016.

[52] H. Brendan McMahan and Matthew J. Streeter. Adaptive bound optimization for online convex optimization. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, pages 244–256, 2010.

[53] Arkadi S. Nemirovski and David B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley UK/USA, 1983.

[54] A.S. Nemirovskii. Interior point polynomial time methods in convex programming, 2004. Lecture Notes.

[55] AS Nemirovskii. Interior point polynomial time methods in convex programming. *Lecture Notes*, 2004.

[56] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[57] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer, 2004.

[58] Y. E. Nesterov and A. S. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia, 1994.

[59] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, New York, NY, USA, 2014.

[60] Francesco Orabona and Koby Crammer. New adaptive algorithms for online classification. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems 2010.*, pages 1840–1848, 2010.

[61] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[62] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. In *ICML*, 2012.

[63] Holger Rauhut. Compressive sensing and structured random matrices. *Theoretical foundations and numerical methods for sparse recovery*, 9:1–92, 2010.

[64] Benjamin Recht. Embracing the random. `http://www.argmin.net/2016/06/23/hyperband/`, 2016.

[65] Benjamin Recht. The news on auto-tuning. `http://www.argmin.net/2016/06/20/hypertuning/`, 2016.

[66] Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 713–719, New York, NY, USA, 2005. ACM.

[67] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 09 1951.

[68] R.T. Rockafellar. *Convex Analysis*. Convex Analysis. Princeton University Press, 1997.

[69] R. Salakhutdinov and N. Srebro. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In *NIPS*, pages 2056–2064, 2010.

[70] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[71] Shai Shalev-Shwartz. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, The Hebrew University of Jerusalem, 2007.

[72] Shai Shalev-Shwartz, Alon Gonen, and Ohad Shamir. Large-scale convex minimization with a low-rank constraint. In *ICML*, pages 329–336, 2011.

[73] Shai Shalev-Shwartz and Yoram Singer. A primal-dual perspective of online learning algorithms. *Machine Learning*, 69(2-3):115–142, 2007.

[74] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Math. Program.*, 127(1):3–30, 2011.

[75] O. Shamir and S. Shalev-Shwartz. Collaborative filtering with the trace norm: Learning, bounding, and transducing. *JMLR - Proceedings Track*, 19:661–678, 2011.

[76] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *ICML*, 2013.

[77] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *arXiv preprint arXiv:1804.04235*, 2018.

[78] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2960–2968, 2012.

[79] Jasper Snoek, Kevin Swersky, Richard S. Zemel, and Ryan P. Adams. Input warping for bayesian optimization of non-stationary functions. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1674–1682, 2014.

[80] Nathan Srebro. *Learning with Matrix Factorizations.* PhD thesis, Massachusetts Institute of Technology, 2004.

[81] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2004–2012, 2013.

[82] Ambuj Tewari, Pradeep D. Ravikumar, and Inderjit S. Dhillon. Greedy algorithms for structurally constrained high dimensional problems. In *NIPS*, pages 882–890, 2011.

[83] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[84] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando de Freitas. Bayesian optimization in high dimensions via random embeddings. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1778–1784, 2013.

[85] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

[86] Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. Linear convergence with condition number independent access of full gradients. In *Advances in Neural Information Processing Systems*, pages 980–988, 2013.

[87] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, pages 928–936, 2003.