Convergence Analysis of an Adaptive Method of Gradient Descent

David Martínez Rubio

Wadham College

University of Oxford

A thesis submitted for the degree of

MSc in Mathematics and Foundations of Computer Science

Trinity 2017

There is nothing so useless
as doing efficiently that which
should not be done at all.
Peter F. Drucker

Acknowledgements

I would like to thank my supervisors, Dr. Varun Kanade and Dr. Frank Wood for their invaluable guidance and feedback. Special thanks to Varun for his dedication to revising the drafts of this dissertation and for patiently listening to my ideas and actively discussing them with me. I would also like to thank K.A. Gillow of the Mathematical Institute for the creation and generous distribution of the OCIAM class, which I am using to write this document.

My most sincere thanks to my family, specially my parents, my brother and my uncle Felix, for their constant support.

Finally, this dissertation is dedicated to my friends, the old and the new, thanks for all the fun.

Abstract

This dissertation studies the convergence of an adaptive method of gradient descent called Hypergradient Descent. We review some methods of gradient descent and their proofs of convergence for smooth and strongly convex functions. We show that the proof of convergence of Adam, a modern popular method of gradient descent, is incomplete. We derive a multiplicative rule of Hypergradient Descent and justify some choices made by, among other things, proposing a method of gradient descent and proving its convergence. The core of the dissertation is the convergence analysis of an instance of Hypergradient Descent. We prove convergence for quadratic functions and for a simple family of unidimensional functions. We also show that the method diverges if some properties are not assumed. We have implemented the algorithms reviewed in this dissertation and some Hypergradient Descent variants and we have applied them to two large scale optimization problems. We compare the executions and conclude that Hypergradient Descent is a good method in practice, specially because it does not need tuning of the learning rate.

Contents

Chapter 1

Introduction

This dissertation is about a family of adaptive algorithms for performing large scale optimization, that we call Hypergradient Descent. These algorithms belong to the category of the gradient descent methods. Almost every algorithm that belongs to this category needs a parameter as input, called learning rate, and the performance of the optimization process depends heavily on the value of this parameter. Finding an optimal value for the learning rate is often essential and requires a lot of computational, and sometimes human, effort. The class of algorithms we treat in this dissertation are adaptive in the sense that they adapt the value of the learning rate in order to remove the dependency of the optimization process on the initial input. Moreover, the adaptation occurs during the optimization process, exploiting the geometry of the function to obtain an approximation to the best value of the learning rate at each moment. This reduces the time that is needed for solving hard optimization problems.

We begin chapter 2 introducing the context of our problem: unconstrained optimization over convex functions with some restrictions. After the context has been presented we describe three important methods of gradient descent, namely Stochastic Gradient Descent, Nesterov's Accelerated Gradient Descent and Adam. The first two of them are two classical and very important algorithms in the field. We give proofs and rates of convergence for them. The third one is a very modern method that is widely used nowadays in the process of training neural networks. There is a convergence analysis under some quite restrictive assumptions in the original paper where Adam appeared for the first time. We have found that the proof is flawed and we will explain the error at the end of chapter 2. This only implies that the proof is incomplete, we do not claim the algorithm does not converge.

Chapter 3 is the core of the dissertation. In it, we present the derivation of Hypergradient Descent based in [2] and [3]; we argue that the rules presented in the latter are not necessarily optimal and we derive a different set of rules. Then we proceed with the exposition of the main results of this dissertation, that is, we analyse the convergence of a particular instance of Hypergradient Descent. We prove that the algorithm does not converge in a more general context than the one we assume. We also prove convergence for quadratic functions and for a particular set of unidimensional convex functions. This proof sheds some light on the difficulty of the problem for more general functions. It is left for future work to prove or refute the convergence of the method for general L-smooth (2.1.2) and μ-strongly convex (2.1.3) functions.

Finally, in chapter 4 we explain an implementation we have done of some of the algorithms presented in this dissertation and we compare some Hypergradient Descent algorithms with the rest in a couple of large scale optimization problems that arise often in Machine Learning. We conclude that Hypergradient Descent algorithms are good at adapting the learning rate, removing the need to feed the algorithm with an optimal one and they are as good as, in fact slightly better than, the other algorithms.

1.1 Maximum Likelihood Estimation in Linear Regression

We start with a motivating example of a classical problem in Statistics and in Machine Learning that requires optimization of quadratic functions: Linear Regression.

Suppose we have that for each $x \in \mathbb{R}^n$ there is a value $y \in \mathbb{R}$ that we will want to predict and that depends on x through a linear model with noise. In other words

$$y = w_0 + w^T x + \varepsilon$$

where $w_0 \in \mathbb{R}$, $w \in \mathbb{R}^n$. Here $\varepsilon \sim N(0, \sigma^2)$ is a term that models the noise and encapsulates irregularities of some process that in practice does not always really behave linearly. We will train the model using some observations $\{(x_i, y_i)\}_{i=1}^k$ that are given. The aim is to find the best w_0, w that better fit the data in the sense that we explain now.

The points $\{x_i\}_{i=0}^k$ are usually obtained from some distribution over $\mathbb{R}^n$. However, in this example we only focus on the relationship between x_i and y_i. We want to find parameters w_0, w that maximize the probability of the observed $\{y_i\}_{i=0}^k$ given

$\{x_i\}_{i=0}^k, w_0, w, \sigma$. In order to simplify the notation we will relabel the points so $x_i \in \{1\} \times \mathbb{R}^n$ and $w \in \mathbb{R}^{n+1}$ contains w_0 as the first coordinate. We will make the common assumption that the noise in each sample is independent from the others. Therefore we want to find $w \in \mathbb{R}^{n+1}$ to maximize:

$$\begin{aligned} p(y_1, \ldots, y_k \mid x_1, \ldots, x_k, w, \sigma) &= \prod_{i=1}^{k} p(y_i \mid x_i, w, \sigma) \\ &= \prod_{i=1}^{k} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{2\pi\sigma^2}\right)^{k/2} \exp\left(-\sum_{i=1}^{k} \frac{(y_i - w^T x_i)^2}{2\sigma^2}\right). \end{aligned}$$

Since we only want to find the argument w that maximizes this expression, we can get rid off the first factor, that is constant. As it is usual, we can then take logarithms and maximize the resulting expression, since the logarithm is a monotone increasing function. Finally, we can remove the $\frac{1}{2\sigma^2}$ from the expression and find w so we maximize $-\sum_{i=1}^{k}(y_i - w^T x_i)^2$ or equivalently, as it is usually written, we want to find:

$$w^* = \text{argmin}_{w \in R^{n+1}} \sum_{i=0}^{k} (y_i - w^T x_i)^2 \tag{1.1}$$

This problem is known as least squares. Most of the tasks in training Machine Learning models, including neural networks, require a similar optimization task, in which we have to minimize a function depending on some training data over $\mathbb{R}^n$. If the model is suitable for the problem at hand, finding the best parameters will let us predict quite precisely future outputs based on inputs. In the case of linear regression, the prediction is $y' = w_0 + w^T x'$, for a new input x'.

Training a model requires a lot of computational effort and in practice, in an attempt to speed-up the optimization algorithms used for training, it is common to use heuristics for which there are not even theoretical guarantees of finding an approximation of the problem solution. Hypergradient Descent is one of these heuristics. In chapter 3 we will get a better understanding of Hypergradient Descent and in particular we will be able to prove that an instance of Hypergradient Descent converges to the minimum for the problem of least squares.

It is common when working with involved optimization methods to begin by analysing their performance on quadratic functions, as we have done in this dissertation. See for example [23].

We would like to remark that there is a closed form solution for the least squares problem. However in large scale optimization problems, i.e. for large n and large k, gradient descent methods are the only feasible computational procedure.

Chapter 2

Gradient descent methods

2.1 Convexity and gradient descent

The main results of this dissertation are on Convex Optimization. We start with the definitions of convex functions and convex sets in $\mathbb{R}^n$.

Definition 2.1.1. A set $C \subset \mathbb{R}^n$ is convex if it contains all the points in the segment formed by any two of its points. Formally, for all $x, y \in C$ and for all $\lambda \in [0, 1]$ we have

$$(1 - \lambda)x + \lambda y \in C.$$

A function $f : C \to \mathbb{R}$ is convex if its graph is below its chords. Formally, for all $x, y \in C$ and for all $\lambda \in [0, 1]$ it is

$$f((1 - \lambda)x + \lambda y) \leq (1 - \lambda)f(x) + \lambda f(y).$$

A central research in Optimization, Machine Learning and many other areas of Computer Science is the study of fast iterative methods for approximately finding the minimum of convex functions. The most widely used methods of optimization for large scale programs are first-order iterative methods, that is, methods that iteratively optimize a function by using only evaluations of the function and its gradient. These are convenient methods for large scale programs due to their memory and time efficient iterations. They are often highly parallelizable as well.

The convex problems we have mentioned take the form

$$\text{argmin}_{x \in C} f(x) \tag{2.1}$$

for a convex function $f : \mathbb{R}^n \to \mathbb{R}$ and a closed convex set $C \subseteq \mathbb{R}^n$, which is known as the constraint set of the problem. First order methods can be modelled in a black

box fashion. These first order black box algorithms use an oracle that given a point $x \in C$ answers with the pair $(f(x), \nabla f(x))$. The complexity of a first order method is usually measured, given $\varepsilon > 0$ and a norm $\|\cdot\|$, in the number of times it is necessary to use the oracle to obtain a point x that approximates a minimizer of the problem x^* by an error of ε, i.e. a point x such that $\|x - x^*\| < \varepsilon$.

Almost every first-order iterative method with provable convergence guarantees uses at least one of two fundamental algorithmic ideas: gradient descent and mirror descent. Gradient descent takes a primal approach by finding points that decrease the objective at every step, generally moving towards directions opposing the gradient, whereas mirror descent follows a complementary dual approach by finding points whose evaluations are no further than a bound from the optimum and it iteratively improves the bound. We will focus on gradient descent methods for unconstrained optimization.

We will make some assumptions on f that are usual in this context. In particular, we will assume that the first derivative of f is L-Lipschitz continuous. Such functions are called L-smooth. Algebraically, this property is the following:

Definition 2.1.2. A differentiable function is said to be L-smooth if

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

Almost every existent gradient descent method assumes that the objective function is smooth. There is a huge literature on gradient descent methods and their rates of convergence in the case in which the only assumptions on the objective function are convexity and L-smoothness. However, in this dissertation we will adopt a more particular framework. The one we have chosen to work with is usual in Machine Learning or any other discipline that requires solving large scale optimization problems with real data whose behaviour is not very erratic. We will work with functions that are also μ-strongly convex (see definition below). Note that if we add a quadratic regularizer to a smooth function, as it is commonly done in Machine Learning, we guarantee that the resultant function is strongly convex and smooth. So this condition is not very restrictive in practice. For convex functions that are only assumed to be L-smooth, the rates of convergence that can be guaranteed are immensely much slower than those in our framework. In particular, regular gradient descent needs $O(1/\varepsilon)$ steps to obtain a solution with an error of at most ε. There is a lower bound of $\mathcal{O}(1/\sqrt{\varepsilon})$ on the number of queries to the oracle that are needed to obtain a solution that approximates a minimizer of a general L-smooth function with

an error of at least ε. There are algorithms whose complexity in the worst case are precisely $\mathcal{O}(1/\sqrt{\varepsilon})$. Nesterov's Accelerated Gradient Descent [16], of which we will talk later, Linear Coupling [1] and Geometric Gradient Descent [5] are some examples. For most methods, at least for the deterministic ones, the rates of convergence in our framework will only need a logarithmic number of steps with respect to $1/\varepsilon$. In the literature, this rate of convergence is called linear, in the sense that the number of queries to the oracle needed is linear with respect to the size of the input ε. We will prove in this section the rates of convergence for some of these methods.

Definition 2.1.3. A function f is μ-strongly convex, for $\mu > 0$, if it is differentiable and satisfies

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

Note that in the case in which f is a twice differentiable function, the assumption of L-smoothness and μ-strong convexity is equivalent to say that if λ is an eigenvalue of the Hessian of f then $0 < \mu \leq \lambda \leq L$.

Recently, in some contexts the hypothesis of strong convexity has been replaced by a weaker hypothesis: the Polyak-Łojasiewicz (PL) inequality. It is weaker in the sense that an L-smooth function that is μ-strongly convex, satisfies the PL inequality with parameter $\frac{4\mu^2}{L}$. Several classical methods and some recent ones have been analysed [13] assuming the PL inequality instead of strong convexity. Surprisingly, the proofs are often simpler; we will make use of it to simplify our exposition.

Definition 2.1.4 (Polyak-Łojasiewicz inequality)**.** Assume $f : \mathbb{R}^n \to \mathbb{R}$ has a non empty solution set C^* for the problem (2.1) and denote by f^* the optimal function value. A function satisfies the PL inequality if the following holds for some $\mu > 0$,

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*),$$

for all $x \in \mathbb{R}^n$.

There are key qualitative differences between μ-strong convexity and the PL inequality. Strong convexity implies that the minimizer is unique. This is not true for functions that satisfy the PL inequality. In fact, functions that satisfy the PL inequality are not necessarily convex.

The following proof is an adaptation of [13] (Appendix A). See the original source for more general results.

Proposition 2.1.5. *Let f be an L-smooth, μ-strongly convex function. Then f satisfies the PL inequality with constant $\frac{\mu^2}{4L}$.*

Proof. Let x^* be the minimizer of f. By the definition of μ-strong convexity the following holds:

$$f^* \geq f(x) + \langle \nabla f(x), x^* - x \rangle + \frac{\mu}{2}\|x^* - x\|^2.$$

Using $f^* - f(x) \leq 0$ and rearranging we obtain

$$\langle \nabla f(x), x - x^* \rangle \geq \frac{\mu}{2}\|x^* - x\|^2.$$

We can bound the left hand side by using Cauchy-Schwartz, divide both sides by $\|x^* - x\|$ and we get

$$\|\nabla f(x)\| \geq \frac{\mu}{2}\|x^* - x\|.$$

Now using the definition of L-smoothness with x^* and x, taking into account that $\nabla f(x^*) = 0$ and using the previous inequality we have the inequality

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

from which we finally obtain the PL inequality.

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \frac{\mu^2}{4L}(f(x) - f^*).$$

□

We will start with the most simple method of gradient descent, which is usually called just gradient descent. This algorithm starts with a point $x_0 \in \mathbb{R}^n$ and it iteratively moves against the direction of the gradient at that point a distance that is proportional to the norm of the gradient. This will guarantee monotone improvement and convergence. The following proof is taken from [13].

Theorem 2.1.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ L-smooth and assume it satisfies the PL inequality with parameter μ. Assume the solution set of problem (2.1) is non empty and let f^* be the minimum value of f. The gradient descent algorithm with $\alpha = \frac{1}{L}$, i.e., with update rule*

$$x_{t+1} = x_t - \alpha \nabla f(x_t), \tag{2.2}$$

converges and it has the following linear convergence rate

$$f(x_t) - f^* \leq \left(1 - \frac{\mu}{L}\right)^t (f(x_0) - f^*).$$

Proof. By using the smoothness condition with x_{t+1} and x_t and using the update rule we obtain

$$f(x_{t+1}) - f(x_t) \leq -\frac{1}{2L}\|\nabla f(x_t)\|^2.$$

Using the PL inequality, the latter is bounded by $-\frac{\mu}{L}(f(x_t) - f^*)$. Rearranging and subtracting f^* from both sides we get

$$f(x_{t+1}) - f^* \leq \left(1 - \frac{\mu}{L}\right)(f(x_t) - f^*).$$

The result follows applying this last inequality recursively. □

The rest of the section is devoted to the exposition of other important gradient descent methods and their convergence proofs. We have implemented these methods and we show in the last section the results of several experiments optimizing some common functions that arise in Machine Learning. We also implemented some of them using the Hypergradient Descent approach. The comparison between these experiments will give an idea of how the method behaves in practice. We first present Stochastic Gradient Descent. The proof of its convergence is an adaptation of the one that appears in [13]. We note that the proof had a small error that we have corrected, the original version did not take expectations with respect to the right variables. After this, we present one of the most influential methods in the field, and the first one to achieve an optimal rate of convergence for L-smooth functions: Nesterov's Accelerated Gradient Descent. We proof convergence in the case that the function is L-smooth and μ-strongly convex. Nesterov's is a method of theoretical importance that gives ideas of how these methods usually work and how their proofs of convergence are done. But also, it serves as an example of a method for which Hypergradient Descent cannot be applied, at least not directly. The proof of convergence is an adaptation of the one in [5]. Finally, we present Adam, a modern (2015) popular adaptive method that is used nowadays in practice to perform large scale optimization. The original paper [14] contains an appendix with a proof that under some very restricted assumptions shows that if the method does not diverge, then it converges and its rate is optimal. It is a surprise that such a popular method does not have a strong theoretical analysis that at least guarantees convergence. In addition, we have found an error in this proof. In the subsection devoted to Adam, we will present the method and we will follow an adaptation of the proof to the point where the error occurs in order to explain it.

2.2 Stochastic Gradient Descent

Stochastic gradient methods apply to the stochastic optimization problem

$$\operatorname{argmin}_{x\in\mathbb{R}^n} f(x) = \operatorname{argmin}_{x\in\mathbb{R}^n} \mathbb{E}_{i\in I}[f_i(x)], \tag{2.3}$$

The main application of these methods is the optimization of finite sums

$$f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x).$$

In Machine Learning, f_i usually represents the fit of a model on an individual training example. These methods are suitable when the number of training examples is so big that the computation of the gradient of f in each iteration requires too many resources. The update rule of stochastic gradient methods is

$$x_{t+1} = x_t - \alpha_t \nabla \tilde{f}_t(x_t) \tag{2.4}$$

where $\nabla \tilde{f}_t(x_t)$ is an randomized estimation of $\nabla f(x_t)$ such that $\mathbb{E}[\nabla \tilde{f}_t(x_t)] = \nabla f(x_t)$. In the case above in which f is a sum of functions we can let $\nabla \tilde{f}_t(x_t)$ be $\nabla f_{i_t}(x_t)$, where $1 \leq i_t \leq m$ is chosen uniformly at random. However, it is more common to estimate the gradient with the so called minibatches. These consist of taking a set J_t of samples of fixed size uniformly at random and define $\nabla \tilde{f}_t := \frac{1}{j}\sum_{j\in J} \nabla f_j(x_t)$. The next theorem will prove convergence of this method. We will use $\nabla \tilde{f}_t(x_t) = \nabla f_{i_t}(x_t)$ only to simplify the notation. Note that the proof is the same for any $\nabla \tilde{f}_t(x_t)$ that satisfies $\mathbb{E}[\nabla \tilde{f}_t(x_t)] = \nabla f(x_t)$.

Theorem 2.2.1. *In the context of problem (2.3), assume that each f_i is L-smooth, f has a non-empty solution set, f satisfies the PL inequality and $\mathbb{E}[\|\nabla f_{i_t}(x_t)^2\|] \leq D^2$ for all x_t and some D. If we iteratively use the update rule (2.4) with $\alpha_t = \frac{2t+1}{2\mu(t+1)^2}$, then we get a convergence rate of*

$$\mathbb{E}[f(x_t) - f^*] \leq \frac{LD^2}{2t\mu^2}.$$

Proof. By using the update rule in the L-smoothness inequality we obtain

$$f(x_{t+1}) \leq f(x_t) - \alpha_t \langle \nabla f(x_t), \nabla f_{i_t}(x_t)\rangle + \frac{L\alpha_t^2}{2}\|\nabla f_{i_t}(x_t)\|^2.$$

Taking expectations with respect to $I_t := \{i_0, \ldots, i_t\}$ we get

$$\begin{aligned}
\mathbb{E}_{I_t}[f(x_{t+1})] &\leq \mathbb{E}_{I_{t-1}}\left[f(x_t) - \alpha_t \langle \nabla f(x_t), \mathbb{E}_{i_t}[\nabla f_{i_t}(x_t)]\rangle + \frac{L\alpha_t^2}{2}\mathbb{E}_{i_t}[\|\nabla f_{i_t}(x_t)\|^2]\right] \\
&\leq \mathbb{E}_{I_{t-1}}\left[f(x_t) - \alpha_t\|\nabla f(x_t)\|^2\right] + \frac{LD^2\alpha_t^2}{2} \\
&\leq \mathbb{E}_{I_{t-1}}\left[f(x_t) - 2\mu\alpha_t(f(x_t) - f^*)\right] + \frac{LD^2\alpha_t^2}{2}.
\end{aligned}$$

In the second inequality we have used $\mathbb{E}_{i_t}[\nabla f_{i_t}(x_t)] = \nabla f(x_t)$ and $\mathbb{E}_{i_t}[\|\nabla f_{i_t}(x_t)\|^2] \leq D^2$. We have used the PL inequality in the third line. Subtracting f^* from both sides gives us:

$$\mathbb{E}_{I_t}[f(x_{t+1}) - f^*] \leq (1 - 2\mu\alpha_t)\mathbb{E}_{I_{t-1}}[f(x_t) - f^*] + \frac{LD^2\alpha_t^2}{2}. \tag{2.5}$$

Substituting $\alpha_t = \frac{2t+1}{2\mu(t+1)^2}$ we obtain

$$\mathbb{E}_{I_t}[f(x_{t+1}) - f^*] \leq \frac{t^2}{(t+1)^2}\mathbb{E}_{I_{t-1}}[(f(x_t) - f^*] + \frac{LD^2(2t+1)^2}{8\mu^2(t+1)^4}.$$

Multiplying both sides by $(t+1)^2$ it is

$$(t+1)^2\mathbb{E}_{I_t}[f(x_{t+1}) - f^*] - t^2\mathbb{E}_{I_{t-1}}[f(x_t) - f^*] \leq \frac{LD^2(2t+1)^2}{8\mu^2(t+1)^2} \leq \frac{LD^2}{2\mu^2}.$$

Adding up this inequality from $t = 0$ to t we obtain

$$(t+1)^2\mathbb{E}_{I_t}[f(x_{t+1}) - f^*] \leq \frac{LD^2(t+1)}{2\mu^2},$$

where we use by convention that $I_{-1} = \emptyset$. □

Theorem 2.2.2. *Under the conditions of the previous theorem, using a constant learning rate $\alpha < \frac{1}{2\mu}$ we get a convergence rate of*

$$\mathbb{E}[f(x_t) - f^*] \leq (1 - 2\mu\alpha)^k(f(x_0) - f^*) + \frac{LD^2\alpha}{4\mu}.$$

Proof. Applying inequality (2.5), that was proved for the previous theorem, recursively with $\alpha_k = \alpha$ we obtain

$$\begin{aligned}
\mathbb{E}[f(x_{t+1}) - f^*] &\leq (1 - 2\alpha\mu)^t(f(x_0) - f^*) + \frac{LD^2\alpha^2}{2}\sum_{i=0}^{t}(1 - 2\alpha\mu)^i \\
&\leq (1 - 2\alpha\mu)^t(f(x_0) - f^*) + \frac{LD^2\alpha^2}{2}\sum_{i=0}^{\infty}(1 - 2\alpha\mu)^i \\
&= (1 - 2\alpha\mu)^t(f(x_0) - f^*) + \frac{LD^2\alpha}{4\mu},
\end{aligned}$$

where in the last line we have used $\alpha < \frac{1}{2\mu}$ and the limit of the geometric series. □

Note that in practice if we don't need more than a fixed accuracy then the result with a constant step-size is perhaps the more useful strategy.

2.3 Nesterov's Accelerated Gradient Descent

As we said previously, this algorithm was the first algorithm to achieve optimal convergence rate in the case of optimization of L-smooth functions. It is surprising that it does not move from one iteration to the next one by following the exact opposite direction of the gradient. Instead, it uses a strategy to combine the gradient with the direction of previous updates.

The algorithm for a function $f : \mathbb{R}^n \to \mathbb{R}$ that is L-smooth and μ-strongly convex is the following. We will denote by κ the condition number, i.e. $\frac{L}{\mu}$.

Algorithm 2.3.1. *(Nesterov's Accelerated Gradient Descent)*

input: α: *learning rate*
input: f
input: x_0: *Initial parameter vector*
  $t \leftarrow 0$
  $y_0 \leftarrow x_0$
  do
    $t \leftarrow t + 1$
    $x_t = y_{t-1} - \alpha \nabla f(y_{t-1})$
    $y_t = x_t(1 + (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)) - (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)x_{t-1}$
  until *convergence*

Theorem 2.3.2. *Let f be L-smooth and μ-strongly convex. Then Nesterov's accelerated gradient descent with $\alpha = \frac{1}{L}$ satisfies*

$$f(x_t) - f(x^*) \leq \frac{\mu + L}{2} \|x_0 - x^*\|^2 \exp\left(-\frac{t}{\sqrt{\kappa}}\right).$$

Proof. We define μ-strongly convex quadratic functions $\Phi_t, t \geq 0$ by induction as follows:

$$\begin{aligned} \Phi_0(x) &= f(y_0) + \frac{\mu}{2}\|x - y_0\|^2, \\ \Phi_{t+1}(x) &= \left(1 - \frac{1}{\sqrt{\kappa}}\right)\Phi_t(x) \\ &\quad + \frac{1}{\sqrt{\kappa}}\left(f(y_t) + \nabla f(y_t)^\top (x - y_t) + \frac{\mu}{2}\|x - y_t\|^2\right). \end{aligned} \tag{2.6}$$

Intuitively Φ_t becomes a finer and finer approximation to f in the following sense:

$$\Phi_t(x) \leq f(x) + \left(1 - \frac{1}{\sqrt{\kappa}}\right)^t (\Phi_0(x) - f(x)). \tag{2.7}$$

Let's prove it by induction. For $t = 0$, it is $\Phi_0(x) \leq \Phi_0(x)$. For the inductive step, we substitute Φ_t by its definition and using $f(x) \geq f(y_t) + \nabla f(y_t)^\top (x - y_t) + \frac{\mu}{2}\|x - y_t\|^2$

it suffices to prove

$$\left(1-\frac{1}{\sqrt{\kappa}}\right)\Phi_{t-1}(x)+\frac{1}{\sqrt{\kappa}}f(x)\leq f(x)+\left(1-\frac{1}{\sqrt{\kappa}}\right)^{t}(\Phi_0(x)-f(x))$$

which is trivial by the induction hypothesis.

Let v_t be the minimizer of Φ_t, i.e. $\operatorname{argmin}_{x\in\mathbb{R}^n}\Phi_t(x)$. Now we will prove the following by induction:

$$f(x_t)\leq\Phi_t(v_t). \tag{2.8}$$

The base case $t=0$ is trivial since $x_0=y_0$. Using the definition of x_{t+1}, L-smoothness, convexity, and the induction hypothesis, we obtain

$$\begin{aligned}
f(x_{t+1}) &\leq f(y_t)-\frac{1}{2L}\|\nabla f(y_t)\|^2\\
&=\left(1-\frac{1}{\sqrt{\kappa}}\right)f(x_t)+\left(1-\frac{1}{\sqrt{\kappa}}\right)(f(y_t)-f(x_t))\\
&+\frac{1}{\sqrt{\kappa}}f(y_t)-\frac{1}{2L}\|\nabla f(y_t)\|^2\\
&\leq\left(1-\frac{1}{\sqrt{\kappa}}\right)\Phi_t(v_t)+\left(1-\frac{1}{\sqrt{\kappa}}\right)\nabla f(y_t)^\top(y_t-x_t)\\
&+\frac{1}{\sqrt{\kappa}}f(y_t)-\frac{1}{2L}\|\nabla f(y_t)\|^2.
\end{aligned}$$

We claim that the last term of the previous inequality is at most $\Phi_{t+1}(v_{t+1})$, which suffices to conclude the proof of the theorem. In order to prove the claim it will be useful to obtain a simpler form of $\Phi_t(x)$. First note that $\nabla^2\Phi_t(x)=\mu I_n$. It is true for $t=0$ and assuming it is true for $t-1$ we have

$$\nabla^2\Phi_t(x)=\left(1-\frac{1}{\sqrt{\kappa}}\right)\nabla^2\Phi_{t-1}(x)+\frac{1}{\sqrt{\kappa}}\mu I_n=\mu I_n$$

Therefore:

$$\Phi_t(x)=\Phi_t(v_t)+\frac{\mu}{2}\|x-v_t\|^2.$$

Now note that by differentiating (2.6), the definition of Φ_t, and using the above form we have

$$\nabla\Phi_{t+1}(x)=\mu\left(1-\frac{1}{\sqrt{\kappa}}\right)(x-v_t)+\frac{1}{\sqrt{\kappa}}\nabla f(y_t)+\frac{\mu}{\sqrt{\kappa}}(x-y_t).$$

Using $\nabla\Phi_{t+1}(v_{t+1})=0$, we can rearrange the equation above to obtain an expression for v_{t+1}:

$$v_{t+1}=\left(1-\frac{1}{\sqrt{\kappa}}\right)v_t+\frac{1}{\sqrt{\kappa}}y_t-\frac{1}{\mu\sqrt{\kappa}}\nabla f(y_t). \tag{2.9}$$

By evaluating Φ_{t+1} at y_t, using the form of Φ_t and Φ_{t+1}, as well as their original definitions we obtain the following:

$$\begin{aligned}\Phi_{t+1}(v_{t+1}) + \frac{\mu}{2}\|y_t - v_{t+1}\|^2 \\ = \left(1 - \frac{1}{\sqrt{\kappa}}\right)\Phi_t(v_t) + \frac{\mu}{2}\left(1 - \frac{1}{\sqrt{\kappa}}\right)\|y_t - v_t\|^2 + \frac{1}{\sqrt{\kappa}}f(y_t).\end{aligned} \tag{2.10}$$

Using (2.9) we get:

$$\begin{aligned}\|y_t - v_{t+1}\|^2 = \langle y_t - v_{t+1}, y_t - v_{t+1}\rangle \\ = \left(1 - \frac{1}{\sqrt{\kappa}}\right)^2\|y_t - v_t\|^2 + \frac{1}{\mu^2\kappa}\|\nabla f(y_t)\|^2 - \frac{2}{\mu\sqrt{\kappa}}\left(1 - \frac{1}{\sqrt{\kappa}}\right)\nabla f(y_t)^\top(v_t - y_t),\end{aligned}$$

which combined with (2.10) gives us

$$\begin{aligned}\Phi_{t+1}(v_{t+1}) =& \left(1 - \frac{1}{\sqrt{\kappa}}\right)\Phi_t(v_t) + \frac{1}{\sqrt{\kappa}}\left(1 - \frac{1}{\sqrt{\kappa}}\right)\nabla f(y_t)^\top(v_t - y_t) + \frac{1}{\sqrt{\kappa}}f(y_t) \\ &- \frac{1}{2L}\|\nabla f(y_t)\|^2 + \frac{\mu}{2\sqrt{\kappa}}\left(1 - \frac{1}{\sqrt{\kappa}}\right)\|y_t - v_t\|^2.\end{aligned}$$

Note that our claim is true if $v_t - y_t = \sqrt{\kappa}(y_t - x_t)$, since we have that $\Phi_{t+1}(v_{t+1})$ is equal to what we want to bound plus the last summand, which is positive. We will prove the latter by induction. By definition of y_{t+1} we have

$$\begin{aligned}(y_{t+1} - x_{t+1})(\sqrt{\kappa} + 1) = (\sqrt{\kappa} - 1)x_{t+1} - (\sqrt{\kappa} - 1)x_t \Leftrightarrow \\ \sqrt{\kappa}(y_{t+1} - x_{t+1}) = \sqrt{\kappa}x_{t+1} - \big(\sqrt{\kappa} - 1\big)x_t - y_{t+1}\end{aligned}$$

Using the definition of x_{t+1} the latter is equal to

$$\begin{aligned}\sqrt{\kappa}y_t - \frac{\sqrt{\kappa}}{L}\nabla f(y_t) - \big(\sqrt{\kappa} - 1\big)x_t - y_{t+1} &= -\frac{y_t}{\sqrt{\kappa}} + \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa}}v_t - \frac{\mu}{\sqrt{\kappa}}\nabla f(y_t) - y_{t+1} \\ &= v_{t+1} - y_{t+1}\end{aligned}$$

We have used the induction hypothesis in the first equality and the definition of v_{t+1} in the second one.

Finally, we can combine (2.7) and (2.8) to prove the theorem:

$$\begin{aligned}f(x_t) - f(x^*) &\leq \Phi_t(x^*) - f(x^*) \\ &\leq \left(1 - \frac{1}{\sqrt{\kappa}}\right)^t(\Phi_0(x^*) - f(x^*)) \\ &= \left(1 - \frac{1}{\sqrt{\kappa}}\right)^t\left(\frac{\mu}{2}\|x_0 - x^*\|^2 + f(x_0) - f(x^*)\right) \\ &\leq \frac{\mu + L}{2}\|x_0 - x^*\|^2\left(1 - \frac{1}{\sqrt{\kappa}}\right)^t\end{aligned}$$

In the last inequality we have used $f(x_0) - f(x^*) \leq \frac{L}{2}\|x_0 - x^*\|^2$, which is true by L-smoothness.

□

We will see later that Nesterov's Accelerated Gradient Descent is a method that is not compatible with the framework of Hypergradient Descent. We will also compare the execution of this algorithm with respect to others and their Hypergradient variants in the last section.

2.4 Adam

The original theoretical framework in which Adam was presented is online convex optimization. The problem of online convex optimization can be formulated as a repeated game between a player and an adversary. At round t, the player chooses an action x_t from some convex subset C of $\mathbb{R}^n$, and then the adversary chooses a convex loss function f_t. The player aims to ensure that the total loss, $\sum_{t=1}^{T} f_t(x_t)$ is not much greater than the smallest total loss $\min_{x \in C} \sum_{t=1}^{T} f_t(x)$, where in this case the action is fixed for every t. The difference between the total loss and its optimal value for a fixed action is known as the regret, which we denote by R_T.

Zinkevich [25] showed that any algorithm incurs a regret that grows at least as $\sqrt{T}$, if f_t are arbitrary convex functions. He also gave an algorithm that achieves this optimal complexity. It was shown later [11] that if every f_t is μ-strongly convex, then it is possible for the regret to grow as $\log(T)$, but the algorithm must know in advance the value of μ. In [12], the authors present an adaptive algorithm that achieves a regret of $\log(T)$ if the functions are μ-strongly convex, for some μ and $\sqrt{T}$ otherwise without the need of knowing the parameters in advance.

We now present Adam, an adaptive gradient descent method that has gained popularity as an optimizer for gradient descent among the machine learning community. It is adaptive in the sense that the distances and directions between updates are adapted according to a heuristic that tries to learn the geometry, at least locally, of the function that is being optimized. As we mentioned before, there is no strong theoretical analysis of this algorithm. The only one we are aware of is the one that appears in the original paper of Adam [14]. They present a proof of convergence under some very restrictive assumptions, like non divergence. We have found that the proof is flawed. We will follow the proof in order to explain the error.

In the framework of online convex optimization, we will have an oracle that at step t returns the pair $(f_t(x), \nabla f_t(x))$, given an input x. We will assume the functions f_t are convex differentiable functions. In practice we usually have a function $f : \mathbb{R}^n \to \mathbb{R}$ and the randomness comes from the evaluation of this deterministic function at random subsamples or minibatches or arises from inherent function noise.

The algorithm is the following:

Algorithm 2.4.1. *(Adam)*

input: α: *learning rate*
input: $\beta_1, \beta_2 \in [0, 1)$
input: ε: *a small value* > 0 *so last step is always well defined*
input: f: *Stochastic objective*
input: x_0: *Initial parameter vector*
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $t \leftarrow 0$
  while x_t *not converged* **do**
    $t \leftarrow t + 1$
    compute $\nabla f_{t-1}(x_{t-1})$ *//We denote* $\nabla f_{t-1}(x_{t-1})_i$ *the* i^{th} *component of* $\nabla f_{t-1}(x_{t-1})$
    for i **in** *1..n* **do**
      $m_{t,i} \leftarrow \beta_1 m_{t-1,i} + (1 - \beta_1)\nabla f_{t-1}(x_{t-1})_i$
      $v_{t,i} \leftarrow \beta_2 v_{t-1,i} + (1 - \beta_2)\nabla f_{t-1}(x_{t-1})_i^2$
      $\widehat{m}_{t,i} \leftarrow m_{t,i}/(1 - \beta_1^t)$
      $\widehat{v}_{t,i} \leftarrow v_{t,i}/(1 - \beta_2^t)$
      $x_{t,i} \leftarrow x_{t-1,i} - \alpha_t \widehat{m}_{t,i}/(\sqrt{\widehat{v}_{t,i}} + \varepsilon)$ *//*$x_{t,i}$ *is the* i^{th} *component of* x_t
    end for
  end while
  return x_t

We will denote the i^{th} component of $\nabla f_t(x_t)$ by $\nabla f_t(x_t)_i$. In order to simplify the notation we will denote by $g_{t,i}$ the vector $(\nabla f_1(x_1)_i, \nabla f_2(x_2)_i, \ldots, \nabla f_t(x_t)_i)$. If we use the norm $\|\cdot\|$ with no subindex we will be referring to the ℓ_2 norm. We will work with a different learning rate at each step. Namely the learning rate at step t, given α, will be $\alpha_t := \alpha/\sqrt{t}$.

In the following, we will go through the proof of convergence originally proposed in [14]. The paper presents two lemmas before the final theorem. We identify an error in each of the two lemmas and one more in the proof of the theorem. The following is one of the two lemmas.

Claim 2.4.1. *Let* $\beta_1, \beta_2 \in [0, 1)$ *such that* $\gamma := \beta_1^2/\sqrt{\beta_2} < 1$. *Assume* $\|\nabla f_t(x_t)\|_\infty \leq G_\infty$. *Then*

$$\sum_{t=1}^{T} \frac{\widehat{m}_{t,i}^2}{\sqrt{t\widehat{v}_{t,i}}} \leq \frac{2G_\infty \|g_{T,i}\|}{(1-\gamma)^2\sqrt{1-\beta_2}}.$$

If we unwrap the definition of m_t and v_t we obtain that it is equal to

$$m_t = (1-\beta_1)\sum_{j=1}^{t}\beta_1^{t-j}\nabla f_j(x_j), \quad v_t = (1-\beta_2)\sum_{j=1}^{t}\beta_2^{t-j}\nabla f_j(x_j)^2.$$

Therefore

$$\begin{aligned}\frac{\widehat{m}_{t,i}^2}{\sqrt{t\widehat{v}_{t,i}}} &= \frac{\sqrt{1-\beta_2^t}}{(1-\beta_1^t)^2}\frac{\left(\sum_{j=1}^{t}(1-\beta_1)\beta_1^{t-j}\nabla f_j(x_j)_i\right)^2}{\sqrt{t\sum_{k=1}^{t}(1-\beta_2)\beta_2^{t-k}\nabla f_k(x_k)_i^2}} \\ &\leq \frac{\sqrt{1-\beta_2^t}}{(1-\beta_1^t)^2}\sum_{j=1}^{t}\frac{t\left((1-\beta_1)\beta_1^{t-j}\nabla f_j(x_j)_i\right)^2}{\sqrt{t\sum_{k=1}^{t}(1-\beta_2)\beta_2^{t-k}\nabla f_k(x_k)_i^2}} \\ &\leq \sum_{j=1}^{t}\frac{t\left(\beta_1^{t-j}\nabla f_j(x_j)_i\right)^2}{\sqrt{t(1-\beta_2)\beta_2^{t-j}\nabla f_j(x_j)_i^2}} = \frac{t}{\sqrt{t(1-\beta_2)}}\sum_{j=1}^{t}\gamma^{t-j}|\nabla f_j(x_j)_i|\end{aligned}$$

In the last inequality we have used the fact that $\sqrt{1-\beta_2^t} < 1$; $(1-\beta_1)^2 < (1-\beta_1^t)^2$ and we have decreased the denominator of each fraction.

If we sum for every $1 \leq t \leq T$ we obtain

$$\sum_{t=1}^{T}\frac{\widehat{m}_{t,i}^2}{\sqrt{t\widehat{v}_{t,i}}} \leq \sum_{t=1}^{T}\frac{|\nabla f_t(x_t)_i|}{\sqrt{(1-\beta_2)}}\sum_{j=0}^{T-t}\sqrt{t+j}\gamma^j,$$

but in the original proof they assert that the right hand side should be

$$\sum_{t=1}^{T}\frac{|\nabla f_t(x_t)_i|}{\sqrt{t(1-\beta_2)}}\sum_{j=0}^{T-t}t\gamma^j, \tag{2.11}$$

and they continue saying that, given that $\sum_{j=0}^{t} j\gamma^j < \frac{1}{(1-\gamma)^2}$, then they can bound the latter by

$$\frac{1}{(1-\gamma^2)\sqrt{1-\beta_2}}\sum_{t=1}^{T}\frac{|\nabla f_t(x_t)_i|}{\sqrt{t}} \leq \frac{1}{(1-\gamma^2)\sqrt{1-\beta_2}}2G_\infty\|g_{T,i}\|. \tag{2.12}$$

The previous bound is wrong. Firstly, because we have to bound $\sum_{j=0}^{T-t} t\gamma^j$ instead of $\sum_{j=0}^{T-t} j\gamma^j$. But even if they have correctly bounded this term using a different argument than what it seems they explain, the last step is not true in general. As a counterexample, take $T = 10$, $|\nabla f_t(x_t)_i| = \frac{1}{2}$, for $1 \leq t \leq T$ and some i, $G_\infty = \frac{1}{2}$. Then

$$\sum_{t=1}^{10}\frac{|\nabla f_t(x_t)_i|}{\sqrt{t}} = \frac{1}{2}\sum_{t=1}^{10}\frac{1}{\sqrt{t}} > \frac{1}{2}\sqrt{10} = 2G_\infty\|g_{T,i}\|.$$

The last step in (2.12) is precisely the other lemma in the original proof. We note that the error the authors make in this one is in the last line (cf. [14], Lemma 10.3), in which they implicitly use

$$-\frac{|\nabla f_t(x_t)_i|}{\sqrt{T}} + \frac{\nabla f_t(x_t)_i^2}{\sqrt{T}} \leq 0.$$

which is not true if $|\nabla f_t(x_t)_i| < 1$.

The following claim is the final "theorem" in [14]:

Claim 2.4.2. *Assume the function f_t has bounded gradients $\|\nabla f_t(x_t)\| \leq G_\infty$ and the distance between any x_t generated by Adam is bounded $\|x_n - x_m\|_2 \leq D$, $\|x_n - x_m\|_\infty \leq D_\infty$ for any $n, m \in \{1, \dots, T\}$. Assume that $\beta_1, \beta_2 \in [0, 1)$ satisfy $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$. Let $\alpha_t = \frac{\alpha}{t}$ and let $\lambda \in (0, 1)$. Suppose that the factor in the definition of m_t is $\beta_1 \lambda^t$ instead of β_1. Adam achieves the following guarantee, for all $T \geq 1$.*

$$\begin{aligned} R(T) \leq & \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|g_{T,i}\| \\ & + \sum_{i=1}^{n} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha(1-\beta_1)(1-\lambda^2)}. \end{aligned}$$

Note that $\widehat{v}_{T,i}$ is bounded by $G_\infty^2/(1-\beta_2^t)$ and that $\sum_{i=1}^n \|g_{T,i}\|$ is bounded by nG_∞, so the complexity is optimal.

The proof uses the previous lemma to obtain the second summand of the bound, and we have already seen that it is wrong. We also note that the argument made to derive the first summand is also wrong. In the last step of this argument they want to bound

$$\sum_{i=1}^{n} \frac{(x_{1,i} - x_i^*)^2 \sqrt{\widehat{v}_{1,i}}}{2\alpha_1(1-\beta_1)} + \sum_{i=1}^{n} \sum_{t=2}^{T} \frac{(x_{t,i} - x_i^*)^2}{2(1-\beta_1)} \left(\frac{\sqrt{\widehat{v}_{t,i}}}{\alpha_t} - \frac{\sqrt{\widehat{v}_{t-1,i}}}{\alpha_{t-1}} \right)$$

by using $\|x_t - x^*\|_\infty \leq D_\infty$ and telescoping the result, obtaining

$$\begin{aligned} & \sum_{i=1}^{n} \frac{D_\infty^2 \sqrt{\widehat{v}_{1,i}}}{2\alpha_1(1-\beta_1)} + \sum_{i=1}^{n} \sum_{t=2}^{T} \frac{D_\infty^2}{2(1-\beta_1)} \left(\frac{\sqrt{\widehat{v}_{t,i}}}{\alpha_t} - \frac{\sqrt{\widehat{v}_{t-1,i}}}{\alpha_{t-1}} \right) \\ & \quad = \frac{D_\infty^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\widehat{v}_{T,i}} \end{aligned}$$

The problem is that we can only bound the second summand if $\sqrt{\widehat{v}_{t,i}}/\alpha_t - \sqrt{\widehat{v}_{t-1,i}}/\alpha_{t-1} \geq 0$, or equivalently

$$\frac{t^2 v_{t,i}}{1-\beta_2^t} \leq \frac{(t-1)^2 v_{t-1,i}}{1-\beta_2^{t-1}} \iff t^2\big(v_{t-1,i}\beta_2 + (1-\beta_2)\nabla f_t(x_t)_i^2\big) \geq (t-1)^2 v_{t-1,i} \frac{1-\beta_2^t}{1-\beta_2^{t-1}}.$$

Grouping all the terms with $v_{t-1,i}$ we obtain

$$v_{t-1,i}\left(t^2\beta_2 - (t-1)^2\frac{(1-\beta_2^t)}{1-\beta_2^{t-1}}\right)+t^2(1-\beta_2)\nabla f_t(x_t)_i^2 \geq 0.$$

The latter is always true only if what multiplies $v_{t-1,i}$ is positive, since $\nabla f_t(x_t)_i^2$ could be very small at some steps. So it should be

$$(t^2\beta_2-t^2\beta_2^t)-(t^2-2t+1)+(t^2\beta_2^t-2t\beta_2^t+\beta_2^t) \geq 0 \Leftrightarrow t^2(\beta_2-1)+2t(1-\beta_2^t)+(\beta_2^t-1) \geq 0.$$

But the latter is no true, for t big enough since $\beta_2 < 1$.

Chapter 3

Hypergradient Descent

This section is devoted to a scheme of gradient descent methods that when applied to some of the most commonly used methods of gradient descent, like Stochastic Gradient Descent and Adam, seems to improve the time required for optimizing large scale unconstrained problems that often arise in Machine Learning. It is a scheme in the sense that given a gradient descent method of a particular kind, that will be described later, it gives another gradient descent method. The original update rule is kept but it is also used to obtain a heuristic to adapt the learning rate to the geometry of the function at each step.

This scheme is called *Hypergradient Descent* (HD). It was rediscovered and published in [3]. We say rediscovered because in the research for this dissertation the following paper [2] was found. In it, a scheme almost identical to HD is presented, along with some variations. In this section, we will explain the derivation of HD, based on an intuitive idea regarding the maximization of the objective value, or its expectation in the case of SGD, at the next step of optimization. We will describe an additive rule for adapting the learning rate, that is the one described in [3]. We will discuss why the additive rule is not intuitively the best choice for HD, we propose a multiplicative rule for the adaptation of the learning rate, that was designed by the author of this dissertation before we knew of the existence of [2]. This paper also proposes a multiplicative rule for HD and other variations that we will describe.

Although the algorithm of HD was discovered some time ago, there is no theoretical analysis of its convergence that we are aware of. In this section, we analyse HD applied to the deterministic gradient descent algorithm and prove its convergence for quadratic functions. We discuss some of the practical problems that the method has and propose some solutions. The resultant scheme seems to be quite independent of the initial learning rate and in practice it seems to work better than the algorithms

without the HD rules. We present experiments showing a comparison of the executions of some of the gradient descent methods studied in this dissertation and their HD counterparts. We also show how the executions for HD are almost independent of the choice of the initial learning rate. Even if the execution barely depends on the initial learning rate, HD has an extra hyperparameter. However, we argue that for most applications it is not necessary to tune this hyperparameter.

3.1 Derivation of Hypergradient Descent and variants

Here we explain where Hypergradient Descent comes from and propose some variations. We comment as well on the form of a HD variant proposed in [2]. The key idea is to perform gradient descent over the learning rate for performing gradient descent. Assume we have a function $f : \mathbb{R}^n \to \mathbb{R}$ and that we have access to an oracle that given a point $x_t \in \mathbb{R}^n$ returns the evaluation of f at x_t and a noisy estimate of $\nabla f(x_t)$, say $\nabla f(x_t) + \varepsilon_t$, where $\mathbb{E}[\varepsilon_t] = 0$. For the derivation of HD in the deterministic setting $\varepsilon_t = 0$, for all t. Let $X_t = (x_0, x_1, \ldots, x_t)$ be the sequence of all the points visited during the course of optimization and let α_t be the learning rate at step t. We will assume the update rule of a point can be seen as a function $x_t = u(X_{t-1}, \alpha_{t-1}, \varepsilon_{t-1})$. The expected value of the objective value at iteration t is $\mathbb{E}[f(x_t)] = \mathbb{E}[f(u(X_{t-1}, \alpha_{t-1}, \varepsilon_{t-1}))]$. We would like to set $\alpha_{t-1} = \alpha^*_{t-1}$, where α^*_{t-1} is the value that minimizes the previous expectation. We can approximate such a value by performing gradient descent over $\mathbb{E}[f(x_t)]$. If we differentiate the previous expression with respect to α_t we obtain

$$\left.\frac{\partial \mathbb{E}[f \circ u]}{\partial \alpha_{t-1}}\right|_{X=X_{t-1}} = \mathbb{E}\left[\nabla f(x_t) \cdot \frac{\partial u(X_{t-1}, \alpha_{t-1}, \varepsilon_{t-1})}{\partial \alpha_{t-1}}\right] = \mathbb{E}\left[(\nabla f(x_t) + \varepsilon_t) \cdot \nabla_\alpha u_{t-1}\right] \tag{3.1}$$

where we have used the shorthand $\nabla_\alpha u_{t-1} = \frac{\partial u(X_{t-1}, \alpha_{t-1}, \varepsilon_{t-1})}{\partial \alpha_{t-1}}$. The last equality is true if we assume, as it is usual, that ε_t and ε_{t-1} are independent. We cannot compute α^*_{t-1} directly from (3.1). Firstly, because we need to pick a value for α_{t-1} in order to compute $\nabla f(x_t) + \varepsilon_t$ and secondly because in general we cannot compute the expectation on the right hand side. However, this expectation is 0 for the optimal parameter α^*_{t-1}. So, if we knew these values, we could try to approximate α^*_{t-1} by performing a step of gradient descent

$$\alpha_{t-1} = \alpha_{t-2} - \beta(\nabla f(x_t) + \varepsilon_t) \cdot \nabla_\alpha u_{t-1}.$$

If we make the assumption that the optimal learning rate for each iteration does not change much across iterations, we could instead use the rule for optimizing the value of the learning rate at the previous iteration

$$\alpha_{t-1} = \alpha_{t-2} - \beta(\nabla f(x_{t-1}) + \varepsilon_{t-1}) \cdot \nabla_\alpha u_{t-2},$$

and this is something we can compute because we know all these values from the previous iteration. This is the general rule that HD uses for adapting the learning rate. If β is a constant, we call this the additive rule of HD, because the learning rate changes in an additive fashion. This additive rule is the one described in [3]. We will argue that there are better choices than the additive rule.

Note that the problem of not knowing $\nabla f(x_t)$ can also be fixed by computing the partial derivative of $x'_t := x_{t-1} - \alpha_{t-2}\nabla f(x_{t-1})$ with respect to the direction $\nabla_\alpha u_{t-1}$ to obtain an approximation to the right term with a little more computational effort. During the research conducted in [3] we took this approach in the first place, but then we suggested and tried the other approach and we saw that, at least in practice, the result of optimization of both is more or less the same, while the first approach that was explained requires less computation.

Let's compute an example of a method of HD. We will use the regular gradient descent update rule (2.2). The value of $\nabla_\alpha u_{t-2}$ is $-\nabla f(x_{t-2})$ and therefore in this case the HD update rules are:

$$x_t = x_{t-1} - \alpha_{t-1}\nabla f(x_{t-1}), \quad \alpha_t = \alpha_{t-1} + \beta\nabla f(x_t)\nabla f(x_{t-1}).$$

Note that for better readability we have shifted the indices of the update rule for the learning rate by 1.

The update rule of HD for α has a geometric interpretation. For a point on the line $g_{t-2}(\alpha) := x_{t-2} - \alpha\nabla f(x_{t-2})$ we have that the gradient of f at $g_{t-2}(\alpha)$ is perpendicular to $\nabla f(x_{t-2})$ if and only if the value of $f(g_{t-2}(\alpha))$ is optimal on the line, and therefore the was no better alpha to improve the local optimization at step $t-2$. Similarly, the dot product between $\nabla f(g_{t-2}(\alpha))$ and $\nabla f(x_{t-2})$ is negative (resp. positive) if α is less than (resp. greater than) an optimal value for α in the line. The update rule for gradient descent in this case increases or decreases α for the next step if it had to be greater or lower at the current one, hoping that the optimal value of α does not change much across iterations.

This geometrical interpretation tells us that the gradient should change in one direction or the other, but in the derivation of HD descent we obtained a precise amount for this change. However, this quantity was picked quite arbitrarily. It is

not clear, in principle, why β should be a constant. In regular gradient descent we have that the distance from one point and the one computed in the next iteration is $\alpha\|\nabla f(x_{t-1})\|$, i.e. it is something that depends on the norm of the gradient and a constant. This is what we have done for the additive rule of HD, we subtract a quantity proportional to the norm of the derivative $\frac{\partial(f\circ u)}{\partial\alpha_{t-1}}$. In the context of gradient descent it makes sense to do it because assuming that the function is L-smooth and that we are at a point x_t, we can guarantee improvement and the maximal improvement that can be guaranteed occurs when we move against the gradient a distance of $\frac{1}{L}\nabla f(x_t)$, for any x_t. However in HD, the process of optimizing α_t is analogous as the one of optimizing $f(g_t(\alpha))$, and this function is $L\|\nabla f(x_t)\|^2$-smooth, which suggests that β should not be a constant and at step t, it should instead be proportional to $\frac{1}{\|\nabla f(x_t)\|^2}$ or something along those lines. In fact it should be something close to $\frac{1}{L\|\nabla f(x_t)\|^2}$. This is better understood with the method of gradient descent we present now, that uses this idea along with the looking ahead rule to improve monotonically the objective value. The proof of convergence suggests that β should also be proportional to α and making this assumption the proof arises naturally. It makes sense that β is proportional to the learning rate since for an L-smooth function we want α to be close to $1/L$, and in this case we have just argued that we want β to be proportional to $1/L$ (and $1/\|\nabla f(x_t)\|^2$). Similarly, if the function f is not L'-smooth but it is in some big enough region, the HD algorithm will try to make α close to $1/L'$, and we can apply the same reasoning as before to see that in this region a good value of β is proportional to $1/L'$. So β depending linearly on α seems a good choice. The following, of course, is only a simplification in which the learning rate adaptations are not accumulated across iterations, which lets us guarantee monotone convergence. However it is useful as an illustrative example.

The algorithm is the following:

Algorithm 3.1.1. *(HD variation)*

input f, x_0, α
$t \leftarrow 0$
repeat
  $t \leftarrow t + 1$
  $\beta \leftarrow \frac{\alpha}{\|\nabla f(x_{t-1})\|^2}$
  $y_t \leftarrow x_{t-1} - \alpha \cdot \nabla f(x_{t-1})$
  $\alpha' \leftarrow \alpha + \beta \cdot \nabla f(y_t) \cdot \nabla f(x_{t-1})$
  $x_t \leftarrow x_{t-1} - \alpha' \cdot \nabla f(x_{t-1})$
until *convergence*

Theorem 3.1.2. *Assume $f : \mathbb{R}^n \to \mathbb{R}$ is L-smooth and satisfies the PL inequality with parameter μ, assume f has a non empty solution set and let f^* be the optimal value. The gradient descent method above with $\alpha = \frac{1}{L}$ converges and it has the following linear convergence rate*

$$f(x_t) - f^* \leq (f(x_0) - f^*) \left(1 - \frac{\mu}{L}\right)^t \prod_{i=1}^{t} \left(1 - \frac{\mu \cos^2(\varphi_i)}{L}\right).$$

where φ_i is the angle that the vectors $\nabla f(x_{i-1})$ and $\nabla f(y_i)$ form.

Proof. By L-smoothness we have

$$f(y_t) - f(x_{t-1}) \leq \langle \nabla f(x_{t-1}), y_t - x_{t-1} \rangle + \frac{L}{2L^2} \|\nabla f(x_{t-1})\|^2 \leq -\frac{1}{2L} \|\nabla f(x_{t-1})\|^2, \tag{3.2}$$

and

$$f(x_t) - f(y_t) \leq \langle \nabla f(y_t), x_t - y_t \rangle + \frac{L}{2} \|x_t - y_t\|^2$$

On the other hand we have

$$x_t - y_t = -\frac{\nabla f(y_t) \cdot \nabla f(x_{t-1})}{L \|\nabla f(x_{t-1})\|^2} \nabla f(x_{t-1}) = -\frac{\|\nabla f(y_t)\| \cos(\varphi_i)}{L \|\nabla f(x_{t-1})\|} \nabla f(x_{t-1})$$

Combining the last two expressions we obtain

$$\begin{aligned} f(x_t) - f(y_t) &\leq -\frac{\|\nabla f(y_t)\| \cos(\gamma)}{L \|\nabla f(x)\|} \langle \nabla f(y_t), \nabla f(x) \rangle + \frac{\|\nabla f(y_t)\|^2 \cos^2(\gamma)}{2L} \\ &\leq -\frac{\cos^2(\gamma)}{2L} \|\nabla f(y_t)\|^2. \end{aligned} \tag{3.3}$$

Now applying the PL inequality to (3.2) and (3.3) we get

$$f(y_t) - f(x_{t-1}) \leq -\frac{\mu}{L}(f(x_{t-1}) - f^*), \quad f(x_t) - f(y_t) \leq -\frac{\mu \cos^2(\gamma)}{L}(f(y_t) - f^*).$$

Rearranging and subtracting f^* to both equation gives us

$$f(y_t) - f^* \leq \left(1 - \frac{\mu}{L}\right)(f(x_{t-1}) - f^*), \quad f(x_t) - f^* \leq \left(1 - \frac{\mu \cos^2(\gamma)}{L}\right)(f(y_t) - f^*).$$

and combining them we have the following, which applied iteratively proves the result

$$f(x_t) - f^* \leq \left(1 - \frac{\mu}{L}\right)\left(1 - \frac{\mu \cos^2(\varphi_t)}{L}\right)(f(x_{t-1}) - f^*).$$

□

Note that the algorithm seems to use two gradients of f in every iteration, which seems to suggest that the method is worse that regular gradient descent since with two steps of gradient descent we can guarantee that

$$f(x_t) - f^* \leq \left(1 - \frac{\mu}{L}\right)^2 (f(x_{t-2}) - f^*),$$

which is strictly better than the guarantee progress that we obtain with one step of the previous algorithm. However, it is not necessary to compute $\nabla f(y_t)$ since we only want $\nabla f(y_t) \cdot \nabla f(x_{t-1})$, which can be seen as a partial derivative of f at the point y_t with respect to the direction of $\nabla f(x_{t-1})$ and multiplied by $\|\nabla f(x_{t-1})\|$. There are efficient methods and software packages that compute this partial derivative in $O\left(\frac{1}{n}\right)$ of the time that is needed to compute the whole gradient [9]. Recall that n is the dimension of the domain of f. So this method can indeed perform much better, with respect to the computational time, than gradient descent. The main purpose of showing this algorithm was only to support our argument against the additive rule of HD, so we will not study in more detail the possible advantages of this method over gradient descent.

As alternative, we propose the following modification in the update rule for the learning rate

$$\alpha_t = \alpha_{t-1}\left(1 - \beta' \frac{\nabla \tilde{f}(x_{t-1}) \cdot \nabla_\alpha u_{t-2}}{\left\|\nabla \tilde{f}(x_{t-1})\right\| \left\|\nabla_\alpha u_{t-2}\right\|}\right). \tag{3.4}$$

Here we have denoted by $\nabla \tilde{f}(x_{t-1})$ to the noisy estimate of $\nabla f(x_{t-1})$ and β' is a constant. This update just makes β proportional to α_{t-1} and to the inverse of the norms in the dot product. The only difference between this rule and what we had mentioned before is that β is proportional to the inverse of the normalization constant in the dot product instead of $\left\|\nabla \tilde{f}(x_{t-1})\right\|^2$. This should not matter too much. We decided to do it in this way because in such a case the method is invariant under rescaling, which is a desirable property for a gradient descent method. We call this rule the multiplicative rule of HD.

If we apply this rule to gradient descent as we did before with the additive one, we obtain the following algorithm. Our convergence analysis will be performed over this algorithm. We use β for the hyperparameter instead of β', we had only used β' before to make clear the relation between this constant and the constant in the additive rule.

Algorithm 3.1.3. *(Multiplicative Hypergradient Descent)*

```
input f : ℝ^n → ℝ
input x_0, α_0, β
t ← 0
repeat
    t ← t + 1
    x_t ← x_{t−1} − α_{t−1} ∇f(x_{t−1})
    α_t ← α_{t−1} ( 1 + β ∇f(x_t)·∇f(x_{t−1}) / (‖∇f(x_t)‖ ‖∇f(x_{t−1})‖) )
until convergence
```

Note that in the case $n = 1$ the two last lines in the body of the loop are equivalent to

```
x_t ← x_{t−1} − α_{t−1} f′(x_{t−1})
if f′(x_t) f′(x_{t−1}) > 0
    α_t ← α_{t−1} (1 + β)
else if f′(x_t) f′(x_{t−1}) < 0
    α_t ← α_{t−1} (1 − β)
```

If we think about HD with multiplicative rule a posteriori, we see that it also has a very convenient property: If at some step t the optimal learning rate is $\bar{\alpha}_t$ and it does not change much in the next few executions, the learning rate α_t will change geometrically until it reaches $\bar{\alpha}_t$ whereas with the additive rule the learning rate changes linearly. In principle, geometric adaptation is much faster. It is also more precise when we want to approximate small values. Note that this does not imply that the multiplicative rule has less precision when adapting large values, since the adaptation also depends on the value of the normalized dot product.

In [2], the authors have proposed a variation of HD. Coincidentally, they also suggest a HD method based on a multiplicative rule, but they do not give any justification for this choice besides what we have just said about the geometric adaptation. They say "We know from our extensive experience with the batch mode adaptive step sizes procedure that it is convenient to adapt step sizes in a geometric way". The authors only do it for gradient descent and momentum, which is other method of gradient descent. Their approach consists of using a learning rate for each dimension, and adapting each of them with a HD rule. They choose a different way to normalize the dot product that appears in HD. The choice seems a bit arbitrary. The whole algorithm is the following:

Algorithm 3.1.4. *(Hypergradient Descent proposed in [2])*

```
input f : ℝ^n → ℝ
input A_0: Diagonal matrix with initial learning rate parameters
input x_0, β
```

input $\gamma \in [0,1)$ – *A parameter generally close to* 1 *used to computer the normalizer factor*
$v_0 = (0,\dots,0)$ – *Each component of* $v_t \in \mathbb{R}^n$ *is a normalizer factor*
$t \leftarrow 0$
repeat
  $t \leftarrow t+1$
  $x_t \leftarrow x_{t-1} - A_{t-1}\nabla f(x_{t-1})$
  for i **in** $1..n$
    $v_{t,i} \leftarrow \gamma v_{t-1,i} + (1-\gamma)\nabla f(x_{t-1})_i^2$
    $\alpha_{t,i} \leftarrow \alpha_{t-1,i}\left(1+\beta\frac{\nabla f(x_t)_i \nabla f(x_{t-1})_i}{v_{t,i}}\right)$ *//*$\alpha_{t,i}$ *is the* i^{th} *element in the diagonal of* A_t.
until *convergence*

There is no previous theoretical analysis of HD in the literature. But it is worth mentioning the work in [18], [19] and [7]. The authors study the convergence of two methods that adapt the learning rate and have some similarities with HD. They only study the convergence of those adaptive algorithms restricting that the learning rate be less than $\frac{1}{L}$ and in the stochastic case. Unfortunately, their techniques and insights are not applicable to the problem studied in this dissertation, in which we study the deterministic case and we do not restrict the learning rate.

3.2 Convergence analysis

In this analysis we will consider HD with the multiplicative rule applied to deterministic gradient descent, i.e. Algorithm 3.1.3, that was presented at the end of the previous section. We will prove that the algorithm does not converge if f is not L-smooth, we will prove convergence in the case of quadratic functions, starting with the ones in 1 dimension, and we prove the convergence of the algorithm for one more 1-dimensional function. The proof of the latter is quite involved, which suggests that the problem of proving convergence in the general case of L-smooth, μ-strongly functions is not easy, not even in the 1-dimensional case.

Proposition 3.2.1. *Algorithm 3.1.3 can diverge if* f *is not* L*-smooth.*

Proof. Take $f(x) = x^4$, which clearly is not L-smooth, for any L. Assume $\beta < \frac{8}{9}$. Then if for some t we have that $|x_t| \geq \sqrt{\frac{1}{\alpha_t}}$ then

$$|x_{t+1}| = |x_t - \alpha_{t-1}\nabla f(x_t)| = |x_t(1-4\alpha_{t-1}x_t^2)| \geq 3|x_t|,$$

$$\alpha_{t+1} = \alpha_t\,(1-\beta) \geq \frac{\alpha_t}{9} \geq \frac{1}{9x_t^2} \geq \frac{1}{x_{t+1}^2}.$$

The last inequality implies $|x_{t+1}| \geq \sqrt{\frac{1}{\alpha_{t+1}}}$. Hence if we start with $|x_0| \geq \sqrt{\frac{1}{\alpha_0}}$ the norm of the objective increases at every step of the algorithm and thus it diverges. □

In the following, given a function f and hyperparameter β we will identify the execution of Algorithm 3.1.3 with the points and learning rates $\{(x_i, \alpha_i)\}_{i=0}^{\infty}$ that would be computed starting in x_0 with initial learning rate α_0. If the algorithm finds the optimizer x^* of f in a finite number of points, say $x_T = x^*$, we will also consider that the sequence is infinite and that $x_i = x^*, \alpha_i = \alpha_T$ for $i > T$. Note that we are not making explicit the dependence of the sequence on f and β, since this will be always clear from the context. Whenever we refer to an arbitrary sequence $\{(x_i, \alpha_i)\}_{i=0}^{\infty}$, we will be referring to one in which $x_0 \in \mathrm{dom}\,(f)$ and $\alpha_0 > 0$, that is, the initial values will be in the set of those for which the algorithm is well defined.

We will denote by x_i^j to be the j^{th} component of x_i.

Lemma 3.2.2. *Let $0 < \mu < L$ be real constants, let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable, μ-strongly convex and L-smooth function and let $\{(x_i, \alpha_i)\}_{i=0}^{\infty}$ be the sequence generated by algorithm 3.1.3. If we have that $\alpha_i > \frac{1}{\mu}$ and $x_i \neq x^*$, then $\alpha_{i+1} < \alpha_i$. Similarly, if $\alpha_i < \frac{1}{L}$ and $x_i \neq x^*$, then $\alpha_{i+1} > \alpha_i$.*

Proof. We will prove that $\alpha_i > \frac{1}{\mu}$ implies $\alpha_{i+1} < \alpha_i$. This is true if and only if $\nabla f(x_{i+1})\nabla f(x_i) < 0$. The other case is analogous.

Let $g(y) := f\left(x_i - y\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|}\right)$. The second derivative of g is

$$g''(y) = \nabla^2 f\left(x_i - y\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|}\right)\frac{-\nabla f(x_i)}{\|\nabla f(x_i)\|}\frac{-\nabla f(x_i)}{\|\nabla f(x_i)\|} = \nabla^2 f\left(x_i - y\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|}\right).$$

Therefore g is μ-strongly convex and it has a unique minimum, let y^* be its minimizer. Due to the assumption $x_i \neq x^*$, the value of $g'(0) = -\|\nabla f(x_i)\|$ must be negative. Since g is convex and $g'(0) < 0$ it must be that the minimizer y^* is greater than 0. We have $\nabla f(x_{i+1})\nabla f(x_i) < 0$ if and only if $y^* \in (0, \alpha_i\|\nabla f(x_i)\|)$. Assume the latter is not true, so $y^* \geq \alpha_i\|\nabla f(x_i)\|$. Since $g'(y^*) = 0$, by the Mean Value Theorem there is $y \in (0, y^*)$ such that

$$g''(y) = \frac{g'(0) - g'(y^*)}{-y^*} = \frac{\|\nabla f(x_i)\|}{y^*} \leq \frac{\|\nabla f(x_i)\|}{\alpha_i\|\nabla f(x_i)\|} < \mu.$$

But this contradicts the fact that g is μ-strongly convex. □

Corollary 3.2.3. *Let f be a function as in Lemma 3.2.2, with $n = 1$. Let I be the interval $\left(\frac{1}{L}, \frac{1}{\mu}(1+\beta)\right)$. If $x_0 \neq x^*$ and $\alpha_0 \notin I$ then there is $n \in \mathbb{Z}^+$ such that $\alpha_n \in I$, $\alpha_i \notin I$ for $i < n$. Also, for $0 < i < n$, if $\alpha_0 < \frac{1}{L}$ then $\alpha_{i+1} = \alpha_i(1+\beta)$ and if $\alpha_0 > \frac{1}{\mu}(1+\beta)$ then $\alpha_{i+1} = \alpha_i(1-\beta)$.*

Proof. It is a straightforward consequence of Lemma 3.2.2 noting that the learning rate changes by a factor of $1-\beta$ or $1+\beta$ if $f'(x_t)f'(x_{t-1})$ is positive or negative respectively. We only have to prove that $x_i \neq x^*$ for $0 \leq i < n$. Assuming that $x_{i-1} \neq x^*$ and using the lemma we get that $\alpha_{i-1} \neq \alpha_i$ and thus $f'(x_i) \neq 0$ which implies $x_i \neq x^*$. Since $x_0 \neq x^*$ the result follows by induction.

We should clarify why I was defined in a way that seems so unnatural. The definition used ensures that the learning rate cannot jump over the entire interval I in a single update. □

Our final goal is to prove the convergence of Algorithm 3.1.3 for quadratic functions. We start proving the one dimensional case, whose ideas are crucial for the general case. A one dimensional quadratic function has an important property that makes the convergence analysis quite simple: the value that the learning rate α_t needs to take so the point x_{t+1} is the minimizer of f is the same for every possible value of x_t. Moreover, if the value of α_t is close to this optimal value, the point x_{t+1} gets close to the minimizer of f.

Theorem 3.2.4. *Given a unidimensional convex quadratic function $f(x) = ax^2 + bx + c$, $a > 0$, Algorithm 3.1.3 converges for any x_0, α_0 and for $\beta < 1$.*

Proof. The algorithm is invariant under translations, so we can assume without loss of generality that $f(x) = ax^2$. Note that

$$\nabla f(x_t)\nabla f(x_{t-1}) > 0 \iff x_t x_{t-1} > 0$$

and that each side is 0 whenever the other is 0. Also, $f(x_t) > f(x_{t-1}) \iff |x_t| > |x_{t-1}|$. The key idea is that in this case for any t we can describe the qualitative behaviour of one step of the algorithm based only on the value of α_{t-1}, for any t:

- $\frac{1}{a} < \alpha_{t-1}$. By definition we have that $x_t = x_{t-1}(1 - 2a\alpha_{t-1})$ and therefore $|x_t| > |x_{t-1}|$. Also, $\alpha_t = \alpha_{t-1}(1-\beta)$ by lemma 3.2.2.

- $\frac{1}{2a} < \alpha_{t-1} \leq \frac{1}{a}$. In this case we also have that $x_t x_{t-1} < 0$ and thus $\alpha_t = \alpha_{t-1}(1-\beta)$ but now $|x_t| \leq |x_{t-1}|$. In particular, if $\alpha_{t-1} \in \left(\frac{1}{2a}, \frac{1}{2a}(1+\beta)\right)$ then

$$|x_t| = |x_{t-1}(1 - 2a\alpha_{t-1})| \leq \beta |x_{t-1}|.$$

- $\alpha_{t-1} < \frac{1}{2a}$. In this case $\alpha_t = \alpha_{t-1}(1+\beta)$, because $x_t x_{t-1} > 0$. Also $|x_t| < |x_{t-1}|$. In particular, if $\alpha_{t-1} \in \left(\frac{1}{2a}(1-\beta), \frac{1}{2a}\right)$ then

$$|x_t| = |x_{t-1}(1 - 2a\alpha_{t-1})| \leq \beta |x_{t-1}|.$$

- $\alpha_{t-1} = \frac{1}{2a}$. In this case $x_t = 0$ and we have convergence.

So if the algorithm starts with $\alpha_0 > \frac{1}{a}$, the value of α_t will decrease exponentially, with ratio $1-\beta$ until the first t_0 such that $\alpha_{t_0} < \frac{1}{2a}$. The norm of x_t will increase whenever $\alpha_{t-1} > \frac{1}{a}$. At this point, for all $t > t_0$ the value of α_t will always be in the interval $\left(\frac{1}{2a}(1-\beta), \frac{1}{2a}(1+\beta)\right)$. This is because, by the previous analysis, whenever $\alpha_{t-1} \in \left(\frac{1}{2a}, \frac{1}{2a}(1+\beta)\right)$ then $\alpha_t = \alpha_{t-1}(1-\beta) > \frac{1}{2a}(1-\beta)$. Similarly, whenever $\alpha_{t-1} \in \left(\frac{1}{2a}(1-\beta), \frac{1}{2a}\right)$ we have that $\alpha_t = \alpha_{t-1}(1+\beta) < \frac{1}{2a}(1+\beta)$. So the value of x_t decreases exponentially with ratio β for $t > t_0$ therefore we have convergence. In the case that $\alpha_{t-1} = \frac{1}{2a}$ the algorithm converges in one step.

In the case that $\alpha_0 < \frac{1}{2a}(1-\beta)$ the learning rate will increase until it is also in $\left(\frac{1}{2a}(1-\beta), \frac{1}{2a}(1+\beta)\right)$ and therefore the algorithm also converges.

We see that the rate of convergence in the worst case depends not only on the initial point x_0 but also on the initial value of the learning rate α_0. In the case $\alpha_0 < \frac{1}{2a}(1+\beta)$, the norm of the objective function does not increase in any step and the number of steps needed for the algorithm to find x_t such that $f(x_t)$ has norm less than ε is

$$t_\varepsilon = \mathcal{O}\Big((1 + \log_{1+\beta}(1/(2a\alpha_0))) + \log_\beta(|x_0|^2/\varepsilon)\Big),$$

where the first summand comes from the number of steps needed for the learning rate to reach the interval $\left(\frac{1}{2a}(1-\beta), \frac{1}{2a}(1+\beta)\right)$ and the second one comes from the steps needed for decreasing the norm of the objective.

The analysis in the case $\alpha_0 > \frac{1}{2a}(1+\beta)$ is similar, but now the norm of the objective increases until the learning rate is less than $\frac{1}{a}$. Let t_0 be the number of steps required for the algorithm to adjust the learning rate such that it is in $\left(\frac{1}{2a}(1-\beta), \frac{1}{2a}(1+\beta)\right)$ for the first time. It is easy to see that the value of t_0 is $\mathcal{O}\big(\log_{1-\beta}(2a\alpha_0)\big)$. We can bound $|x_{t_0}|$ by using $|x_t| \leq (2a\alpha_0 - 1)|x_{t-1}|$ for all $t \leq t_0$. We conclude that in this case, given $\varepsilon > 0$, the number of steps t_ε needed for the algorithm to find $f(x_t)$ with norm less than ε is

$$t_\varepsilon = \mathcal{O}\Big(\big(\log_{1-\beta}(2a\alpha_0) + \log_\beta\big(|x_0^2|(2a\alpha_0 - 1)^{2t_0}/\varepsilon\big)\big)\Big).$$

□

In order to shed light on the difficulty of the problem of determining if Algorithm 3.1.3 converges or not in the general case for an L-smooth and μ-strongly convex function, we proceed to prove convergence when the function f consists of two distinct halves of parabolas spliced at its minimum. This function is quite simple and yet the

proof of convergence is fairly involved. This suggests that the problem, even in the one dimensional case, has certain complexity. The proof that follows leverages one of the key properties of Algorithm 3.1.3: in the one dimensional case, after one decrement and one increment of the learning rate, the result is smaller than the original learning rate.

Theorem 3.2.5. *Let $0 < \mu < L$ be real constants. Let $f : \mathbb{R}^n \to \mathbb{R}$ be the function $f(x) = \frac{1}{2}Lx^2$ for $x \leq 0$ and $f(x) = \frac{1}{2}\mu x^2$ for $x > 0$. Given f, Algorithm 3.1.3 converges for any x_0, α_0 and for β small enough.*

Proof. If $x_i > 0$ then $x_{i+1} > 0$ if and only if $\alpha_i < \frac{1}{\mu}$. Similarly if $x_i < 0$ then $x_{i+1} < 0$ if and only if $\alpha_i < \frac{1}{L}$. Using corollary 3.2.3, it suffices to prove convergence when $\alpha_0 \in \left(\frac{1}{L}, \frac{1}{\mu}(1+\beta)\right)$. In such a case if $x_0 \leq 0$ then $x_1 \geq 0$. Also, if $x_i > 0$ and $\alpha_i < \frac{1}{\mu}$, we obtain $x_{i+1} > 0$ and $\alpha_{i+1} = \alpha_i(1+\beta) < \frac{1}{\mu}(1+\beta)$. Thus at some point t_0, for the first time, $\alpha_{t_0} \in \left(\frac{1}{\mu}, \frac{1}{\mu}(1+\beta)\right)$ and $x_{t_0} > 0$. It is possible that the algorithm converges before reaching this point, only in the case that the learning rate takes the value $\frac{1}{\mu}$ exactly.

We will prove that if we have $x_t > 0$ and $\alpha_t \in \left(\frac{1}{\mu}, \frac{1}{\mu}(1+\beta)\right)$ after a finite number of steps r, bounded by a constant C, $x_{t+r} > 0$, $\alpha_{t+r} \in \left(\frac{1}{\mu}, \frac{1}{\mu}(1+\beta)\right)$ and $\|x_{t+r}\| \leq D\|x_t\|$ with D a constant less than 1. The constants C and D only depend on f and β. Convergence follows from the previous property.

It must be that $x_{t+1} < 0$. We now have two cases:

1. $x_{t+2} > 0$. This implies $\alpha_{t+2} = \alpha_t(1-\beta)^2$ and now for some steps, say r, the points x_{t+2+i} will be positive and the learning rate will increase until it surpasses $\frac{1}{\mu}$. Since $\frac{1}{\mu} < \alpha_t(1+\beta)^r(1-\beta)^2 < \frac{1}{\mu}(1+\beta)$ and $\frac{1}{\mu} < \alpha_t < \frac{1}{\mu}(1+\beta)$ the value of r is s or $s+1$, where s is the greatest number such that $(1+\beta)^s(1-\beta)^2 < 1$, i.e.,
$$s = \left\lfloor -\frac{2\log(1-\beta)}{\log(1+\beta)} \right\rfloor \geq 2. \tag{3.5}$$
Also, for all $0 < i < s$, we have that $\frac{1}{\mu}(1-\beta)^2 < \alpha_t(1-\beta)^2 < \alpha_{t+2+i}$ so
$$|x_{t+2+i+1}| = |x_{t+2+i}|\left(1 - \alpha_{t+2+i}\mu\right) \leq |x_{t+2+i}|\left(1-(1-\beta)^2\right) \leq 2\beta|x_{t+2+i}|.$$
Let $0 < \delta < 1$. Using the former s times we obtain the following bound on the absolute value of x_{t+r+2}:
$$\begin{aligned}|x_{t+r+2}| \leq |x_{t+s+2}| \leq (2\beta)^s|x_{t+2}| &= (2\beta)^s|1-L\alpha_{t+1}||1-\mu\alpha_t||x_t| \\ &\leq (2\beta)^s\left|1-\frac{L}{\mu}\right|\beta|x_t| \leq (2\beta)^{s+1}\left|1-\frac{L}{\mu}\right||x_t| \overset{?}{<} \delta|x_t|.\end{aligned}$$

We have used that $x_{t+2} = (1 - L\alpha_{t+1})(1 - \mu\alpha_t)x_t$. Also, the inequality between the last term in the first line and the first term in the second line uses $\frac{1}{L} < \alpha_{t+1} < \frac{1}{\mu}$ and $\alpha_t \in \left(\frac{1}{\mu}, \frac{1}{\mu}(1+\beta)\right)$. The inequality with the question mark is true if

$$\beta < \frac{1}{2}\sqrt[s+1]{\frac{\delta}{\frac{L}{\mu} - 1}}. \tag{3.6}$$

2. $x_{t+2} < 0$. This can only happen if $\alpha_t(1-\beta) < \frac{1}{L}$, and the latter only happens if $\frac{1}{\mu}(1-\beta) < \frac{1}{L}$. Suppose that for some i we have that $\alpha_i \in \left(\frac{x_i}{f'(x_i)}(1-\beta)^2, \frac{x_i}{f'(x_i)}(1+\beta)\right)$. Note that $\frac{x_i}{f'(x_i)}$ is $\frac{1}{L}$ if $x_i < 0$ and $\frac{1}{\mu}$ if $x_i > 0$. It must be:

$$|x_{i+1}| = |x_i|\left|1 - \alpha_i\frac{f'(x_i)}{x_i}\right| \leq \left(2\beta - \beta^2\right)|x_i| \leq 2\beta|x_i|.$$

Since $x_{t+1}, x_{t+2} < 0$. The learning rate increases in step $t+2$, i.e. $\alpha_{t+2} > \alpha_{t+1}$. It will keep increasing until for some k we have $x_{t+k} > 0$ for the first time. In that case $\alpha_{t+k} = \alpha_{t+k-1}(1-\beta) < \frac{1}{\mu}(1+\beta)(1-\beta) < \frac{1}{\mu}$, whereas $\alpha_{t+k-1} > \frac{1}{L}$. Since now $x_{t+k} > 0$ and $\alpha_{t+k} < \frac{1}{\mu}$, the learning rate will increase again successively until the first step $t+r$, included, such that $\alpha_{t+r} > \frac{1}{\mu}$. We have that α_t and α_j are in $\left(\frac{1}{\mu}(1-\beta)^2, \frac{1}{\mu}(1+\beta)\right)$, for $t+k < j < t+r$ because

$$\alpha_{t+k+1} = \alpha_{t+k-1}(1+\beta)(1-\beta) \geq \alpha_{t+k-1}(1-\beta) \geq \frac{1}{L}(1-\beta) > \frac{1}{\mu}(1-\beta)^2,$$

and $\alpha_{j+1} > \alpha_j$. We also have that x_t and x_j are positive. Also, $\alpha_j \in \left(\frac{1}{L}(1-\beta), \frac{1}{L}(1+\beta)\right)$, for $t+1 \leq j \leq t+k$. Hence $|x_{t+r}| \leq (2\beta)^r|x_t|$. During this process, the learning rate only decreased twice.

We started with $\alpha_t > \frac{1}{\mu}$ and we have finished with $\alpha_{t+r} > \frac{1}{\mu}$. That means that $(1+\beta)^{(r-2)-1}(1-\beta)^2 < 1$, because otherwise $\alpha_{t+r-1} \geq \alpha_t > \frac{1}{\mu}$. Thus, in this case r is bounded by a constant that only depends on f and β in the same way as in the previous case.

Assume β satisfies the inequality (3.6) and is less than $\frac{1}{2}$. It may not seem straightforward if this is possible, since the definition of s involves β. However, but as pointed out in (3.5), $s \geq 2$, so we can always pick

$$\beta < \frac{1}{2}\sqrt[2]{\frac{\delta}{\frac{L}{\mu} - 1}} < \frac{1}{2}\sqrt[s+1]{\frac{\delta}{\frac{L}{\mu} - 1}}.$$

It is clear by the analysis of the previous two cases that we can set $C = s + 1$ and $D = \max((2\beta)^{s+1}, \delta)$ so the aforementioned property holds and the algorithm converges. □

Theorem 3.2.6. *Let A be a diagonal matrix with eigenvalues $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Let $f : \mathbb{R}^n \to \mathbb{R}$ be the function $f(x) = \frac{1}{2}x^T Ax$. Any sequence $\{(x_i, \alpha_i)\}_{i=0}^{\infty}$ computed by Algorithm 3.1.3 converges, given f and $\beta < 1$.*

We present two auxiliary lemmas before the proof. Note that f is λ_n-smooth and λ_1-strongly convex. The fact that f can be decomposed as a sum of quadratic unidimensional functions is a crucial property that will be used in the proof. This is because if we have a point x_i, the behaviour of a component of x_i only depends on the learning rate, regardless of the value of the other components. On the other hand, the behaviour of the learning rate is not quite regular. The proof of Theorem 3.2.6 proceeds by induction assuming that the learning rate is always less than a constant C and then it improves that constant. The following lemma proves that if the learning rate is always less than a constant less than $\frac{2}{\lambda_j}$ then the j^{th} component will converge to zero. The learning rate has to also be lower bounded by a positive constant, but this is always true for every sequence generated by Algorithm 3.1.3 since the learning rate always increases if it is less than $\frac{1}{\lambda_n}(1-\beta)$. We will use this lemma to guarantee convergence of the components whose eigenvalues λ satisfy $C < \frac{2}{\lambda}$, and thus simplifying the problem.

Lemma 3.2.7. *Let f be as in Theorem 3.2.6 and let $s = \{(x_i, \alpha_i)\}_{i=0}^{\infty}$ be a sequence generated by Algorithm 3.1.3. Let $i_0 \in \mathbb{N}$, $j \in \{1, \ldots, n\}$, $0 < \gamma_1, \gamma_2 < 1$. If $(1-\gamma_1)\frac{1}{\lambda_j} < \alpha_i < \frac{1}{\lambda_j}(1+\gamma_2)$ for all $i \geq i_0$ then $\left|x_{i+1}^j\right| < \left|x_i^j\right| \max(\gamma_1, \gamma_2) < \left|x_i^j\right|$ and in particular*

$$\lim_{i\to\infty} x_i^j = 0.$$

Proof.

$$\begin{aligned}\left|x_{i+1}^j\right| &= \left|x_i^j\right||1 - \alpha_i\lambda_j| \leq \left|x_i^j\right| \max\left(\left|1 - (1-\gamma_1)\frac{1}{\lambda_j}\lambda_j\right|, \left|1 - (1-\gamma_2)\frac{1}{\lambda_j}\lambda_j\right|\right) \\ &= \left|x_i^j\right| \max(\gamma_1, \gamma_2) < 1.\end{aligned}$$

The second claim is a straightforward consequence of the first one. □

Let's specify the induction hypothesis that we will use in the proof of the theorem. We will assume that we have convergence if the learning rate is always less than $\frac{1}{\lambda_j}\frac{3+\beta}{2}$ and we will prove convergence in the case in which the learning rate is less than the same expression for the next lower eigenvalue, that is $\frac{1}{\lambda_{j-1}}\frac{3+\beta}{2}$. The magic constant $\frac{3+\beta}{2}$ is just a number less than 2 and greater than $1+\beta$. It is less than 2 to have the convergence of the j^{th} coordinate by the previous lemma. Why it is useful that

the constant is greater than $1+\beta$ is the key of the proof and will be clear later. But basically, it ensures that at every step i in which the learning rate has increased and has become at least $\frac{1}{\lambda_j}\frac{3+\beta}{2}$, we have $\nabla f(x_{i-1})\nabla f(x_i) > 0$ at the same time that the summands of the previous expression that depend on the components whose indices are at least j are negative. On the other hand by the previous lemma we know the rest of the components converge. So at every step i in which this happens, we know that the negative components cannot be very big, since the total sum is positive. If this happened an infinite number of times we could ensure the convergence of the other components obtaining global convergence. If this does not happen the learning rate has to only decrease after some point. The following lemma will cover this case and it will also be useful to reduce the problem to the case $\alpha_0 < \frac{1}{\lambda_1}(1+\beta)$.

Lemma 3.2.8. *Let f be as in Theorem 3.2.6 and let $\{(x_i, \alpha_i)\}_{i=0}^{\infty}$ be a sequence generated by Algorithm 3.1.3. Suppose there is i_0 such that for every $i \geq i_0$ the learning rate only decreases, i.e. $\alpha_{i+1} < \alpha_i$. Let $k = \max\{j \mid x_0^j \neq 0\}$. Then for i big enough $\frac{1}{\lambda_k} < \alpha_i < \frac{2}{\lambda_k}$ and the sequence converges.*

Proof. We can assume without loss of generality that $k = n$ because the analysis of a sequence from some point with some coordinates equal 0 is the same as the analysis we would obtain if we ignored those coordinates. By the same reason, we can assume without loss of generality that $x_{i_0}^j \neq 0$ for every j such that $\lambda_j = \lambda_n$.

Recall that f is λ_n-smooth. Hence by Lemma 3.2.2 it must be $\alpha_i \geq \frac{1}{\lambda_n}$ for all $i \geq i_0$ because otherwise the learning rate would increase. Also, it cannot be $\alpha_i = \frac{1}{\lambda_n}$ because in that case $\nabla f(x_i) \cdot \nabla f(x_{i+1}) = \sum_{j=1}^{n} \lambda_j^2 \left(x_i^j\right)^2 (1 - \alpha_i \lambda_j) \geq 0$ and thus the learning rate at step $i+1$ would not have decreased. Note that $x_i^j \neq 0$ for all j such that $\lambda_j = \lambda_n$ and for all $i \geq i_0$ because it can only be 0 if $\alpha_i = \frac{1}{\lambda_n}$, for some $i \geq i_0$.

Since the sequence of learning rates is bounded there must exist $\alpha := \lim_{i\to\infty} \alpha_i$. This means that the difference between one learning rate and the next one gets arbitrarily small when i increases. Thus $\frac{\nabla f(x_i)\cdot\nabla f(x_{i-1})}{\|\nabla f(x_i)\|\|\nabla f(x_{i-1})\|} \to 0$. There must be the case that for some $j < n$, $\lambda_j < \lambda_n$ because if $\lambda_1 = \lambda_2 = \cdots = \lambda_n$ then $\frac{\nabla f(x_i)\cdot\nabla f(x_{i-1})}{\|\nabla f(x_i)\|\|\nabla f(x_{i-1})\|} = -1$.

If $\alpha < \frac{2}{\lambda_n}$ then there is i_1 such that for all $i \geq i_1$, $\alpha_i < \frac{2}{\lambda_n} \leq \frac{2}{\lambda_j}$, for any j. Convergence follows in this case by Lemma 3.2.7. So it only remains to prove that the case $\alpha \geq \frac{2}{\lambda_n}$ is impossible. As we will see now, the components whose eigenvalues are λ_n grow much faster than the others. We will be able to derive a contradiction based on this. In particular, we will prove that the cosine of the angle between two consecutive gradients tends to -1. If x_i^j grows too much, then the i^{th} coordinates of

two consecutive gradients are also very large and since they will have opposite signs we will be able to argue that the angle that the gradients form is close to π. In this case $\alpha_i > \frac{2}{\lambda_n}$. Let $j < n$ be such that $\lambda_j < \lambda_n$. We have

$$\begin{aligned} |1-\alpha_i\lambda_n| - |1-\alpha_i\lambda_j| &\geq \alpha_i\lambda_n - 1 - \max(\alpha_i\lambda_j - 1, 1-\alpha_i\lambda_j) \\ &> \min\left(\frac{2\lambda_j}{\lambda_n}, 2 - \frac{2\lambda_j}{\lambda_n}\right) > 0. \end{aligned} \tag{3.7}$$

Call $\delta = \min\left(\frac{2\lambda_j}{\lambda_n}, 2 - \frac{2\lambda_j}{\lambda_n}\right)$. We want to prove that for any $\varepsilon > 0$, there is i big enough, such that $\left|x_i^j\right|/|x_i^n| \leq \varepsilon$. So take $\varepsilon > 0$. There is i_1 such that for every $i \geq i_1$ we have $\lambda_n\alpha_i - \lambda_n\alpha \leq \frac{\delta}{2}$, or equivalently $\lambda_n\alpha - \frac{\delta}{2} \geq \lambda_n\alpha_i - \delta$. Hence, given any $\varepsilon' > 0$, there must also be $i_2 \geq i_1$ such that for all $i \geq i_2$

$$\varepsilon' \prod_{k=i_1}^{i} (\alpha_k\lambda_n - 1) \geq \varepsilon' \prod_{k=i_1}^{i} (\alpha\lambda_n - 1) \geq \prod_{k=i_1}^{i} (\alpha\lambda_n - 1 - \frac{\delta}{2}) \geq \prod_{k=i_1}^{i} (\alpha_k\lambda_n - 1 - \delta). \tag{3.8}$$

The second inequality in the previous expression is true because for $a > b$ there is always ℓ big enough, such that $\varepsilon' a^\ell \geq b^\ell$. Now, if we apply (3.8) with $\varepsilon' = \varepsilon\left|x_{i_1}^n\right|/\left|x_{i_1}^j\right|$, there are i_1 and i big enough such that the following holds:

$$\varepsilon|x_i^n| = \varepsilon'\left|x_{i_1}^n\right| \prod_{k=i_1}^{i} |1-\alpha_k\lambda_n| \geq \left|x_{i_1}^j\right| \prod_{k=i_1}^{i} (\alpha_k\lambda_n - 1 - \delta) \geq \left|x_{i_1}^j\right| \prod_{k=i_1}^{i} |1-\alpha_k\lambda_j| = \left|x_i^j\right|,$$

which is what we wanted. The last inequality is true by (3.7) and the definition of δ.

Let $S = \{j \mid \lambda_j < \lambda_n\}$. Let a_i be such that $a_i^j = \nabla f(x_i)^j = \lambda_j x_i^j$ if $j \in S$ and 0 otherwise and let $b_i = \nabla f(x_i) - a_i = Ax_i - a_i$. Note that $b_{i+1} = (1-\alpha_i\lambda_n)b_i$. Given $\varepsilon/|S|$ pick i_1 and i big enough such that for every $j \in S$ and every $k \notin S$ we have $|S|\left|\lambda_j x_i^j\right| \leq \varepsilon\left|\lambda_n x_i^k\right|$, i.e. $|S|\left|a_i^j\right| \leq \varepsilon\left|b_i^k\right|$. In that case we have $\|a_i\| \leq |S| \max_{j\in S}\left|a_i^j\right| \leq \varepsilon\|b_i\|$. Then, distributing and using Cauchy-Schwarz inequality we have

$$\begin{aligned} &\frac{\nabla f(x_i)\cdot\nabla f(x_{i-1})}{\|\nabla f(x_i)\|\|\nabla f(x_{i-1})\|} = \frac{(a_i+b_i)\cdot(a_{i+1}+b_{i+1})}{\|a_i+b_i\|\|a_{i+1}+b_{i+1}\|} \leq \\ &\leq \frac{\|a_i\|\|a_{i+1}+b_{i+1}\| + \|b_i\|\|a_{i+1}\| + (1-\alpha_i\lambda_n)\|b_i\|^2}{\|a_i+b_i\|\|a_{i+1}+b_{i+1}\|} \leq \varepsilon + \varepsilon - \frac{1}{\varepsilon^2+1}. \end{aligned}$$

We have used several inequalities here. Since $a_i \perp b_i$ we have $\max(\|a_i\|, \|b_i\|) \leq \|a_i+b_i\|$. Note that since $|1-\alpha_i\lambda_n| > |1-\alpha_i\lambda_j|$ then $\|a_{i+1}\| \leq |1-\alpha_i\lambda_n|\|a_i\|$ and also $\|a_{i+1}+b_{i+1}\| \leq |1-\alpha_i\lambda_n|\|a_i+b_i\|$. For the bound of the first summand we have used that $\|a_i\| \leq \varepsilon\|b_i\| \leq \varepsilon\|a_i+b_i\|$. For the second one we have used that $\|b_i\| \leq \|a_i+b_i\|$ and

$\|a_{i+1}\| \leq |1-\alpha_i\lambda_n|\|a_i\| \leq \varepsilon|1-\alpha_i\lambda_n|\|b_i\| = \varepsilon\|b_{i+1}\| \leq \varepsilon\|a_{i+1}+b_{i+1}\|$. For the third one we have used

$$\frac{\|b_i\|}{\|a_i+b_i\|} = \frac{1}{\sqrt{\frac{\|a_i\|^2}{\|b_i\|^2}+1}} \geq \frac{1}{\sqrt{\varepsilon^2+1}} \text{ and } \frac{|1-\alpha_i\lambda_n|\|b_i\|}{\|a_{i+1}+b_{i+1}\|} \geq \frac{\|b_i\|}{\|a_i+b_i\|} \geq \frac{1}{\sqrt{\varepsilon^2+1}}.$$

This proves that it is impossible that $\frac{\nabla f(x_i)\cdot\nabla f(x_{i-1})}{\|\nabla f(x_i)\|\|\nabla f(x_{i-1})\|} \to 0$.

□

Proof of Theorem 3.2.6. As we explained before, we will prove by induction on k that if there is i_0 such that $\alpha_i < \frac{1}{\lambda_{n-k+1}}\frac{3+\beta}{2}$ for every $i \geq i_0$, then the sequence converges. In particular, if the sequence starts with $\alpha_0 > \frac{1}{\lambda_1}(1+\beta)$, by Lemma 3.2.2 the learning rate will decrease infinitely or it will be less than $\frac{1}{\lambda_1}(1+\beta)$ at some point. In the former case, we have convergence by Lemma 3.2.8. In the latter case the hypothesis for $k=n$ is true, so the induction proof that follows is sufficient.

The base case is $k=1$. It is a direct consequence of Lemma 3.2.7 since the learning rate is lower bounded by $\min\left(\alpha_0, \frac{1}{\lambda_n}(1-\beta)\right)$. This lower bound is trivial considering Lemma 3.2.2.

Assume the property is true for $k-1 < n$, now we will prove it for k. So $\alpha_i < \frac{1}{\lambda_{n-k+1}}\frac{3+\beta}{2}$ $\left(< \frac{2}{\lambda_{n-k+1}} \leq \frac{2}{\lambda_{n-j+1}}, j \geq k\right)$ for $i \geq i_0$. In a similar fashion as in the base case, by Lemma 3.2.7 it must be that for $j \geq k$ we have $\left|x_i^{n-j+1}\right| \to 0$. By the induction hypothesis there is i_1 such that if $\alpha_i < \frac{1}{\lambda_{n-k+2}}\frac{3+\beta}{2}$ for every $i \geq i_1$ then the sequence converges. Otherwise there is an infinite set of indices I such that $\alpha_i \in \left[\frac{1}{\lambda_{n-k+2}}\frac{3+\beta}{2}, \frac{1}{\lambda_{n-k+1}}\frac{3+\beta}{2}\right)$. In fact, by Lemma 3.2.8, either the sequence converges or there is an infinite subset of indices $I' \subset I$ such that the learning rate has increased. That is, for every $i \in I'$ it is $\alpha_{i-1} < \alpha_i$. Note that

$$\alpha_{i-1} \geq \alpha_i/(1+\beta) > \frac{1}{\lambda_{n-k+2}}\frac{3+\beta}{2(1+\beta)} > \frac{1}{\lambda_{n-k+2}}. \tag{3.9}$$

The fact that $\alpha_{i-1} < \alpha_i$ implies $\nabla f(x_{i-1})\nabla f(x_i) > 0$ or equivalently

$$\sum_{j=1}^{n} \lambda_j^2 \left(x_{i-1}^j\right)^2 (1-\alpha_{i-1}\lambda_j) > 0. \tag{3.10}$$

And here is the key idea of the proof, the idea we exposed before. We have that some of the previous summands are negative, but those that are positive correspond to the coordinates whose convergence we have guaranteed by Lemma 3.2.7. Yet, the sum is positive so the other coordinates must remain small and since I' is infinite, they must converge as we will see now. Let $a_i = \min\{j \mid 1-\alpha_{i-1}\lambda_j < 0\}$ and let

$b_i = \max_{j>a_i}\{x_{i-1}^j\}$. We have just seen in (3.9) that $\alpha_{i-1} > \frac{1}{\lambda_{n-k+2}}$, so $a_i > n-k+2$ and indeed the coordinates whose indices j satisfy $1-\alpha_i\lambda_j > 0$ are some of the coordinates, or all of them, whose convergence we have proved by using Lemma 3.2.7. Hence $b_i \to_{i\to\infty} 0$. If we take the negative summands to the right hand side of (3.10) and divide by $\lambda_{a_i}^2(\alpha_{i-1}\lambda_{a_i}-1)$ we obtain

$$\sum_{j<a_i} \frac{\lambda_j^2\big(x_{i-1}^j\big)^2(1-\alpha_{i-1}\lambda_j)}{\lambda_{a_i}^2(\alpha_{i-1}\lambda_{a_i}-1)} > \sum_{j\geq a_i}^{n} \frac{\lambda_j^2\big(x_{i-1}^j\big)^2(\alpha_{i-1}\lambda_j-1)}{\lambda_{a_i}^2(\alpha_{i-1}\lambda_{a_i}-1)} > \sum_{j\geq a_i}^{n}\big(x_{i-1}^j\big)^2.$$

Now we only have to bound the left hand side. Given $j < a_i$, we have that $\big(x_{i-1}^j\big)^2 \leq b_i^2$, $\lambda_j^2/\lambda_{a_i}^2 < 1$ and $1-\alpha_i\lambda_j < 1$. Also, using (3.9) we have $\alpha_{i-1}\lambda_{a_i} > \alpha_{i-1}\lambda_{n-k+2} > \frac{3+\beta}{2(1+\beta)}$ $\Big(>1\Big)$. So we can upper bound the left hand side by

$$\frac{(n-a_i+1)b_i^2}{(3+\beta)/(2+2\beta)-1}.$$

And because $b_i \to_{i\to\infty} 0$ we have $x_i^j \to_{i\to\infty} 0$ for every j and thus the sequence converges. □

Corollary 3.2.9. *Algorithm 3.1.3 converges for $f(x) = x^TAx + b^Tx + c$, where $A \in \mathcal{M}_{n\times n}$ is a symmetric semidefinite-positive matrix, $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$.*

Proof. On the one hand, we can assume that $c = 0$ since $\operatorname{argmin}\big(x^TAx + b^Tx + c\big) = \operatorname{argmin}\big(x^TAx + b^Tx\big)$. On the other hand, Algorithm 3.1.3 is invariant under isometries and translations of the space. An isometry of the space is given by an orthogonal matrix, and symmetric matrices with real entries are diagonalizable by orthogonal matrices and they have real eigenvalues. Let P be the orthogonal matrix that diagonalizes A, i.e. $P^TAP = D$, with D diagonal. Hence, the algorithm converges if it does it for $f(Px) = x^tDx + b^TPx$. It is easy to see that there is a translation of the space such that the previous function does not have linear term on x, i.e. there is $z \in \mathbb{R}^n$ such that $f(Px+z) = x^TD'x + c'$, where D' is also a semidefinite diagonal matrix and c' is a constant. Finally $\operatorname{argmin}(f(Px+z)) = \operatorname{argmin}(x^TD'x)$. We can ignore the coordinates for which D' has eigenvalues equal to 0, because the gradient of these coordinates is always 0 and thus they do not change. Therefore, we get by Theorem 3.2.6 that Algorithm 3.1.3 converges for this problem and so it does for $f(x)$. □

Corollary 3.2.10. *Algorithm 3.1.3 can be used to solve the problem of least squares 1.1.*

Proof. We can represent the function $f(w) = \sum_{i=1}^{k}(y_i - w^T x_i)^2$ with vector notation. Let $y = (y_1, \ldots, y_k)^T$ and let X be the matrix whose rows are $x_1^T, \ldots, x_k^T$. Then

$$f(w) = w^t X^T X w - 2y^T X w + y^T y$$

And the result follows by Corollary 3.2.9 and the fact that $X^T X$ is symmetric and semidefinite-positive. Both properties are straightforward: $\left(X^T X\right)^T = X^T \left(X^T\right)^T = X^T X$ and $w^T X^T X w = (Xw)^T (Xw) \geq 0$. □

Chapter 4

Experimental results

In this section we apply the methods in the previous sections and some of their HD variants to two problems: fitting a logistic regression classifier and the training of a multi-layer neural network. The aim of this section is to show that HD algorithms slightly outperform the executions of the non HD variants but since they adapt the learning rate there is no need for tuning any hyperparameter. We will start describing the algorithms used and will conclude commenting on the results of the experiments.

4.1 Algorithms

We have implemented Stochastic Gradient Descent 2.4 (SGD), Adam 2.4.1 and their HD variants using the multiplicative rule. Using the definition of HD with multiplicative rule 3.4 we have that the update rule of HD for SGD is

$$\alpha_t \leftarrow \alpha_{t-1}\left(1+\beta\frac{\nabla\tilde{f}_t(x_t)\cdot\nabla\tilde{f}_{t-1}(x_{t-1})}{\left\|\nabla\tilde{f}_t(x_t)\right\|\left\|\nabla\tilde{f}_{t-1}(x_{t-1})\right\|}\right)$$

while the update rule of HD for Adam is:

$$y_{t,i} \leftarrow \widehat{m}_{t-1,i}/(\sqrt{\widehat{v}_{t-1,i}}+\varepsilon), \quad \text{for } 1\leq i\leq n$$
$$\alpha_t \leftarrow \alpha_{t-1}\left(1+\beta\frac{\nabla\tilde{f}_t(x_t)\cdot y_t}{\left\|\nabla\tilde{f}_t(x_t)\right\|\left\|y_t\right\|}\right).$$

Note that HD needs to compute the gradient at the point obtained at each iteration. In most gradient descent methods, we need to compute this gradient anyway, so the extra computation that the HD variants require is quite little. However, in

Nesterov's Accelerated Gradient Descent 2.3.1 at each iteration we do not compute the gradient at the point obtained in each iteration but the gradient of an intermediate point, for which we do not have guarantees of convergence. If we were going to implement the HD variant for Nesterov's algorithm, we would have to compute two gradients at each iteration, making the method much worse in practice than just Nesterov's algorithm. We have implemented Nesterov's method without HD in order to compare its execution to the rest of the algorithms. In fact, we have implemented a slight different method to the one presented before, since we do not know the condition number a priori. There are two alternatives to solve this problem. We could treat the condition number as a hyperparameter and execute the method with different values of it and take the best of the executions. However this requires a lot of extra computation. In his original work, Nesterov also proposed a method that does not need to know the condition number, although the theoretical bounds of convergence are worse, see [5]. We have implemented this version of Nesterov's Accelerated Gradient Descent. This method defines the following numbers:

$$\lambda_0 = 0, \quad \lambda_t = \frac{1 + \sqrt{1 + 4\lambda_{t-1}^2}}{2}, \text{ and } \gamma_t = \frac{1 - \lambda_t}{\lambda_{t+1}}.$$

The update rule is the same as in 2.3.1, but substituting $(\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$ by γ_t, that is:

$$x_t = y_{t-1} - \alpha \nabla f(y_{t-1})$$
$$y_t = (1 - \gamma_t)x_t + \gamma_t x_{t-1},$$

for $t \geq 1$. Recall that the initial point x_0 is an input of the algorithm and that $x_0 = y_0$. In the implementation we do not use the gradient in each iteration but an approximation of it using minibatches.

We saw in the previous section some advantages that the multiplicative rule has versus the additive one. However, in practice, the multiplicative rule presents a problem: if the initial learning rate is very small, the multiplicative distance between this initial learning rate and the optimal learning rate in the beginning could be very large. This will make the algorithm very slow in the beginning until the learning rate finds a good approximation of the optimal learning rate. Solving this problem will make the method almost independent of the choice of the initial learning rate, which is one of the best properties of our final algorithms. The solution is quite simple: when the execution starts, detect if the learning rate has to increase or decrease. If it has to increase (resp. decrease) multiply it by 2 (resp. divide it by 2) successively

until the learning rate has to decrease (resp. increase). If we were multiplying the learning rate by 2 we divide it once by 2. Hence after this process the learning rate is always less than the optimal learning rate at the initial point and the factor that the learning rate has to be multiplied by is at most 2. Then we start the HD algorithm. In this way, with a reasonable value of β, the learning rate will not need a lot of iterations to approximate the optimal learning rate for the first time. We can detect if the learning rate needs to increase or decrease computing x_1 and seeing if the HD rule would have increased the learning rate. Then we update the learning rate and repeat the process using x_0 and the new learning rate. We have defined this process in such a way that we start the algorithm with a learning rate less than the optimal one because otherwise the norm of the objective could increase a lot in the first iterations. Mathematically, this does not present any problem, since after the learning rate has been adapted sufficiently well, the objective will decrease quickly. However, in practice we could incur in overflow errors.

4.2 Experiments

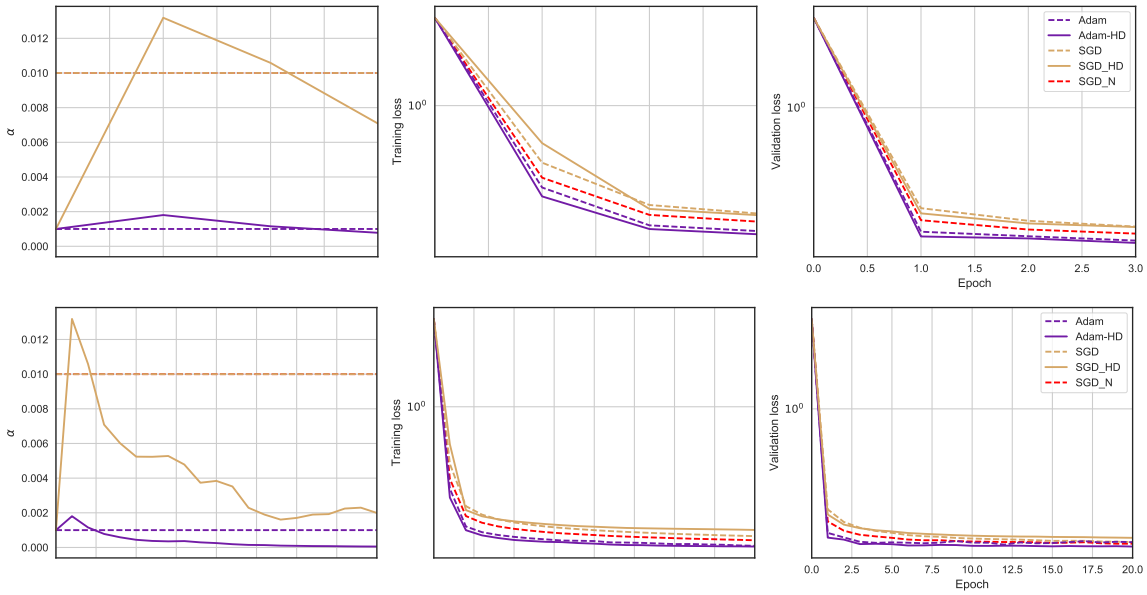
For the experiments we have used the framework and the code used in [3], which uses Torch [6]. In particular we have used the same architectures for solving Logistic Regression and training a Multi-Layer Perceptron (MLP). We have implemented all the HD update rules.

For the logistic regression classifier we have used the MNIST database (`http://yann.lecun.com/exdb/mnist/`) assigning membership probabilities for ten classes to input vectors of length 784. We use the full $60,000$ images in MNIST for training and compute the validation loss using the $10,000$ test images. L2 regularization is used with a coefficient of 10^{-4}. For the algorithms without HD descent, we have tuned the learning rate so the performances are approximately optimal. The resulting values for the learning rate are 10^{-2} for SGD and Nesterov's method and 10^{-3} for Adam.

The MLP neural network is executed again on the MNIST database. The network consists of two fully connected hidden layers with $1,000$ units each and ReLU activations. L2 regularization is applied with a coefficient of 10^{-4}. We also tuned the learning rate for the algorithms without HD and the optimal values are 10^{-2} for SGD and Nesterov's method and 10^{-3} for Adam. We use the default values for $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ which are the values suggested in the original paper [14].

The value of the hyperparameter β indicates the speed of adaptation. We used $\beta = 0.05$ for all the executions. We also note that for most applications this value is

Logistic Regression. Plots showing 3 epochs (1^{st} row) and 20 epochs (2^{nd} row).



MLP training. Plots showing 3 epochs (1^{st} row) and 20 epochs (2^{nd} row).

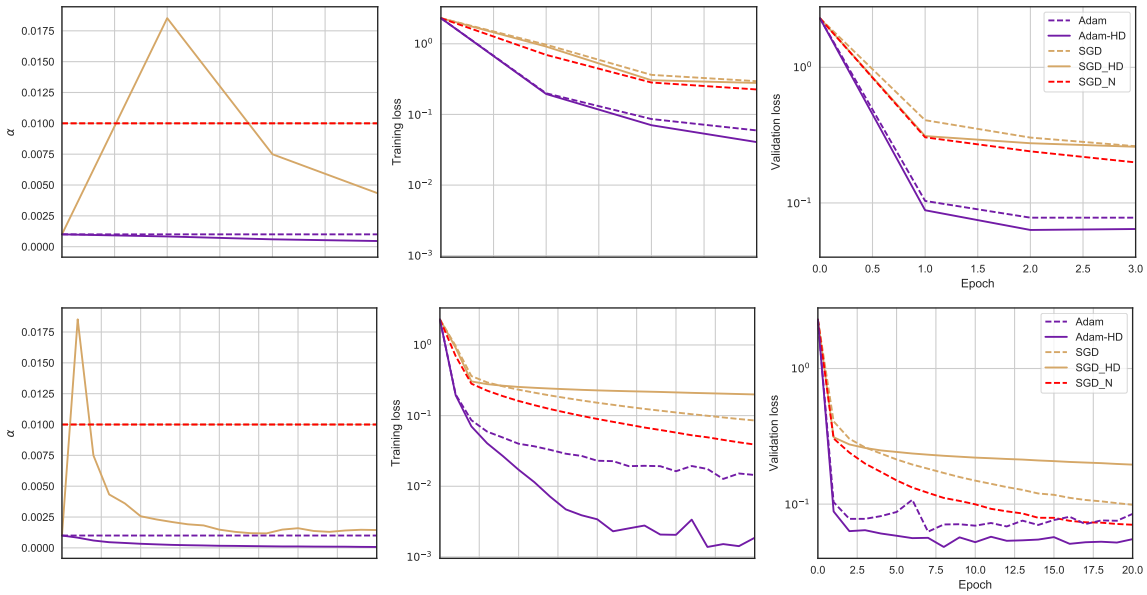


Figure 4.1

good enough because the learning rate adapts exponentially fast and this value only changes the base. That is, the adaptation process is faster than the actual change of the optimal learning rate needed at each step and thus there is no apparent need to tune this base to its optimal value.

In Figure 4.1 we show the execution of 3 and 20 epochs for Linear Regression and MLP with the algorithms SGD, Adam, their HD corresponding algorithms SGD-HD and Adam-HD and finally SGD-N that is Nesterov's method. One epoch is one full pass through the entire training set of $60,000$ images with a minibatch size of 128. In the same figure, we show executions of 3 and 20 epochs of SGD-HD and Adam-HD for Linear Regression and MLP starting with very different learning rates. We see that the learning rate becomes more or less the same for all the cases at any given time and the training loss is also similar. The validation loss was similar as well.

In both problems we can see how the learning rate for SGD-HD, that started with a small value, increases quickly to approximate an optimal learning rate that at the beginning seems to be large and then it adapts little by little to the geometry of the function at every moment. In the case of Adam the learning rate shows a similar behaviour but with much smaller values, mainly due to the fact that the parameters $\widehat{m}_t$, $\widehat{v}_t$ in the Adam algorithm already adapt the distance between one update and the other making value of the learning rate for this algorithm usually lower than for SGD.

The logistic regression problem is convex, but it is not the case for MLP. That is probably the reason why close to epoch 20 each algorithm's loss decreases very slowly, because they are in a local minimum. We have executed the same network for different starting points and the results have been all similar. It seems that SGD-HD always gets stuck in worst local minima than the others algorithms. In any case, in these kinds of non convex problems, our interest should be in the first epochs, and it seems that in this case the HD variants behave a bit better.

However, we are comparing the best execution of the non HD variants against one HD variant. The execution of the non HD variants are much worse with a different initial learning rate. The desirable property that HD offers is that the choice of the initial learning rate is not important, since it will be adapted to the geometry of the problem. In figures 4.3 we show the executions of SGD-HD and ADAM-HD for both problems Logistic Regression and MLP with different initial learning rates: 10^{-6}, 10^{-4}, 10^{-3}, 10^{-2} . We see that the executions are very similar in every case, which is what we expected.

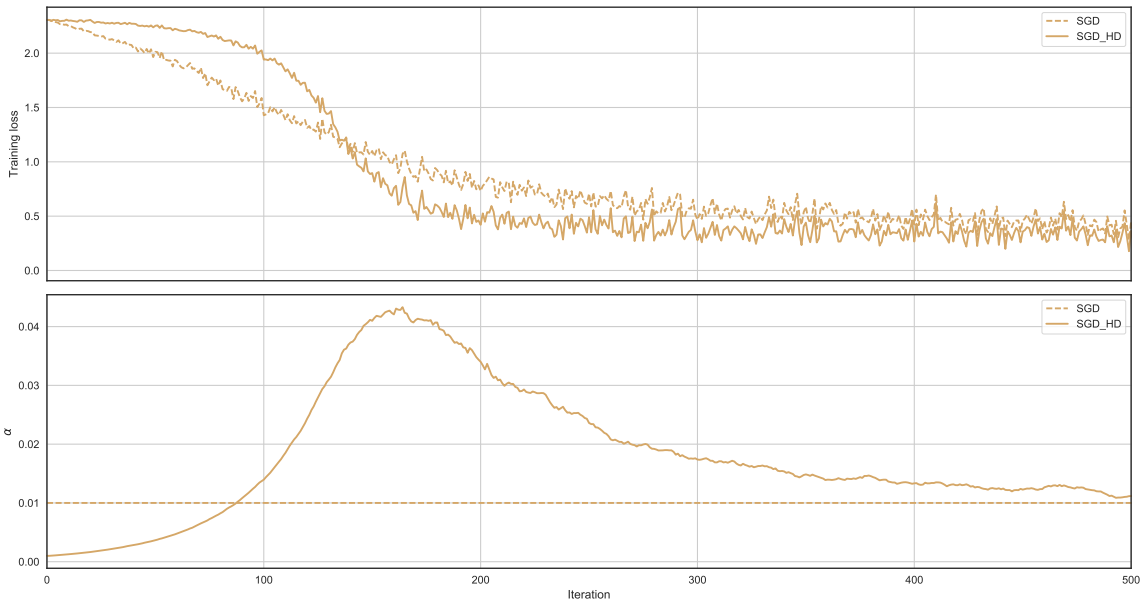


Figure 4.2: SGD adaptation

The results in plots showing loss versus iteration or epoch remain virtually the same when they are plotted versus wall-clock time. This is not unexpected since the extra computation that the HD rules involve is very small.

We conclude this section with the experiment of Figure 4.2 which shows the loss function of the same execution in Figure 4.1 of SGD-HD for MLP after each iteration instead of after each epoch. This plot shows that the behaviour is precisely what we wanted. We start with a small value for the learning rate that is not optimal for the starting point and the SGD-HD algorithm adapts the learning rate increasing it until it is 0.04. The loss function does not decrease so much as the one of the SGD algorithm at the beginning because SGD-HD needs some iterations to approximate well the optimal learning rate for the first time but after it has got a good approximation the loss decreases much faster.

In conclusion, the execution of the HD variants seems to work slightly better than the best execution of the non HD algorithms while at the same time we do not have to tune the initial learning rate of the HD algorithms, making them a convenient option for solving large scale optimization problems.

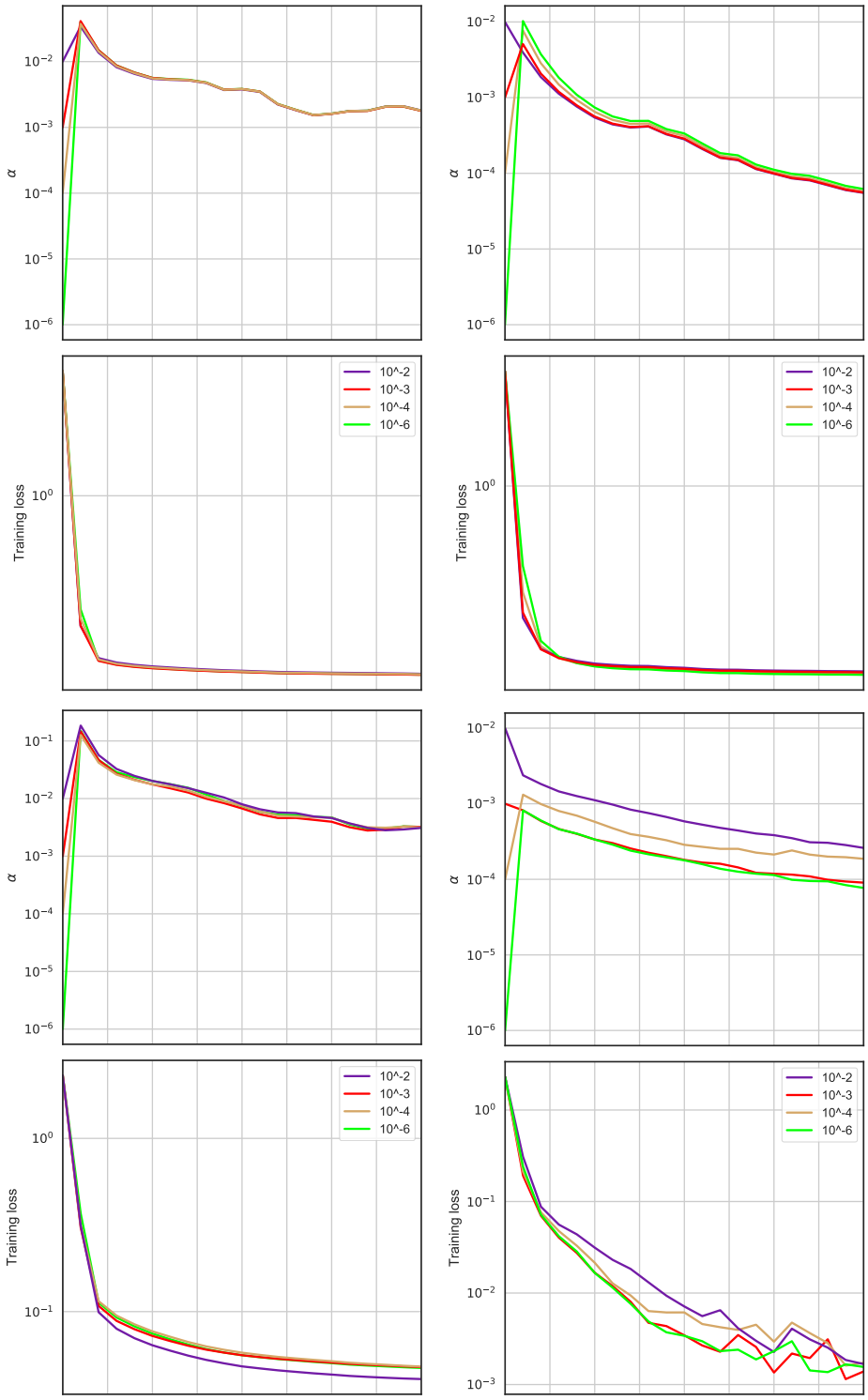


Figure 4.3: SGD (left) and Adam (right) executions with different initial values of the learning rate. The first four plots correspond to Logistic Regression and the four last plots to MLP.

Bibliography

[1] Zeyuan Allen-Zhu and Lorenzo Orecchia, *Linear coupling: An ultimate unification of gradient and mirror descent*, arXiv preprint arXiv:1407.1537 (2014).

[2] Luís B Almeida, Thibault Langlois, José D Amaral, and Alexander Plakhov, *Parameter adaptation in stochastic optimization*, On-Line Learning in Neural Networks, Publications of the Newton Institute (1998), 111–134.

[3] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood, *Online learning rate adaptation with hypergradient descent*, arXiv preprint arXiv:1703.04782 (2017).

[4] Stephen Boyd, Lin Xiao, and Almir Mutapcic, *Subgradient methods*, lecture notes of EE392o, Stanford University, Autumn Quarter **2004** (2003).

[5] Sébastien Bubeck et al., *Convex optimization: Algorithms and complexity*, Foundations and Trends® in Machine Learning **8** (2015), no. 3-4, 231–357.

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet, *Torch7: A MATLAB-like environment for machine learning*, Biglearn, nips workshop, 2011.

[7] João Pedro Antunes Ferreira da Cruz, *Algoritmos de aproximação estocástica com valor do passo adaptativo* (2005).

[8] John Duchi, Elad Hazan, and Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research **12** (2011), no. Jul, 2121–2159.

[9] Andreas Griewank and Andrea Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*, SIAM, 2008.

[10] Moritz Hardt, *The zen of gradient descent*, Mimeo, 2013.

[11] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal, *Logarithmic regret algorithms for online convex optimization*, Colt, 2006, pp. 499–513.

[12] Elad Hazan, Alexander Rakhlin, and Peter L Bartlett, *Adaptive online gradient descent*, Advances in neural information processing systems, 2008, pp. 65–72.

[13] Hamed Karimi, Julie Nutini, and Mark Schmidt, *Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition*, Joint european conference on machine learning and knowledge discovery in databases, 2016, pp. 795–811.

[14] Diederik Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[15] Jayanth Koushik and Hiroaki Hayashi, *Improving stochastic gradient descent with feedback*, arXiv preprint arXiv:1611.01505 (2016).

[16] Yurii Nesterov, *A method of solving a convex programming problem with convergence rate o (1/k2)*, Soviet mathematics doklady, 1983, pp. 372–376.

[17] Brendan O'donoghue and Emmanuel Candes, *Adaptive restart for accelerated gradient schemes*, Foundations of computational mathematics **15** (2015), no. 3, 715–732.

[18] Alexander Plakhov and Pedro Cruz, *A stochastic approximation algorithm with step-size adaptation*, Journal of Mathematical Sciences **120** (2004), no. 1, 964–973.

[19] ———, *A stochastic approximation algorithm with multiplicative step size modification*, Mathematical Methods of Statistics **18** (2009), no. 2, 185–200.

[20] Benjamin Recht, *Cs726-lyapunov analysis and the heavy ball method* (2010).

[21] Tom Schaul, Sixin Zhang, and Yann LeCun, *No more pesky learning rates*, International conference on machine learning, 2013, pp. 343–351.

[22] Weijie Su, Stephen Boyd, and Emmanuel J Candes, *A differential equation for modeling nesterov's accelerated gradient method: theory and insights*, Journal of Machine Learning Research **17** (2016), no. 153, 1–43.

[23] Jialei Wang, Weiran Wang, and Nathan Srebro, *Memory and communication efficient distributed stochastic optimization with minibatch prox*, arXiv preprint arXiv:1702.06269 (2017).

[24] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht, *The marginal value of adaptive gradient methods in machine learning*, arXiv preprint arXiv:1705.08292 (2017).

[25] Martin Zinkevich, *Online convex programming and generalized infinitesimal gradient ascent*, Proceedings of the 20th international conference on machine learning (icml-03), 2003, pp. 928–936.