ABSTRACT

DAY, JOSHUA THOMAS. Online Algorithms for Statistics. (Under the direction of Dr. Hua Zhou and Dr. Eric Laber.)

Traditional algorithms for calculating statistics and models are often infeasible when working with big data. A statistician will run into problems of not just scalability, but of handling data arriving in a continuous stream. Online algorithms, which update estimates one observation at a time, can naturally handle big and streaming data. Many traditional (offline) algorithms have online counterparts that produce exact estimates, but this is not always possible. There exists a variety of online (stochastic approximation) algorithms for approximate solutions, but there is no universally "best" algorithm and convergence can be sensitive to the choice of learning rate, a decreasing sequence of step sizes.

Majorization-minimization (MM) is an optimization concept that has not received much attention in the stochastic approximation (SA) literature. MM is an intuitive idea of iteratively solving easier problems that guarantee a decrease in the objective function that incorporates some second-order information in each iteration. The current state-of-the-art SA algorithms are based entirely on first-order (gradient) information and therefore ignore potentially useful information in each update. We derive two new algorithms for incorporating MM concepts into SA that have strong stability properties. The first algorithm (OMAP) is similar in spirit to the Stochastic MM Algorithm (Mairal, 2013b), referred to here as OMAS. We analyze OMAP and OMAS in a unified framework and offer stronger convergence results for OMAS than in the original paper. The second algorithm (MSPI) incorporates the MM concept into Implicit Stochastic Gradient Descent (Toulis and Airoldi, 2015) and Stochastic Proximal Iteration (Ryu and Boyd, 2014). Compared to the algorithms on which it is based, MSPI can solve a wider class of problems and in many cases has a cheaper online update. For all three MM-based algorithms, it is typically straightforward to translate existing offline MM algorithms into an online counterpart, particularly in the case of quadratic majorizations.

OnlineStats is a Julia package that implements the above algorithms as well as a large catalog of online algorithms for statistics. Using a unifying representation of how statistics/models get updated, a small interface is all that is needed for performant operations that can also be run in parallel. Most software that implements online algo-

rithms typically focus on a small class of problems or specific type of algorithm. To our knowledge, OnlineStats is unique in the way algorithms are represented, the scope of problems it can solve, and the ability to easily add new methods.

Stochastic approximation algorithms are notoriously difficult to compare. They react differently to the type of model, learning rate, etc., and theoretical bounds typically provide little insight into how an algorithm behaves “on average”. We create a novel visualization technique based on simulated data that alleviates many of the difficulties in SA algorithm comparison and use it to examine eight state-of-the-art SA methods, including the two new algorithms introduced in this dissertation. The comparisons are calculated under a variety of conditions: four types of models (two differentiable and two non-differentiable), three learning rates, and different number of parameters in the model.

Online Algorithms for Statistics

by
Joshua Thomas Day

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Statistics

Raleigh, North Carolina

2018

APPROVED BY:

Dr. Len Stefanski	Dr. Alyson Wilson
Dr. Hua Zhou Co-chair of Advisory Committee	Dr. Eric Laber Co-chair of Advisory Committee

DEDICATION

To my parents, who have supported me in every way.

BIOGRAPHY

Josh Day was born on February 10, 1986, in Minneapolis, MN. Before attending NC State, he earned a B.S. in Mathematics/Statistics and B.A. in Economics/Music from Winona State University in Minnesota. Josh enjoys working on difficult optimization problems and translating paper solutions into efficient computer programs (particularly with the Julia language). Besides math and computers, he enjoys music, playing guitar, hiking, biking, and ultimate frisbee.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor, Dr. Hua Zhou, for his mentorship, guidance, and time. I am continually inspired by his work ethic, intelligence, curiosity about what he doesn't know, and dedication to his students. It was from Hua that I first heard about the Julia language, so I thank him for not only that but for the research meetings in which my previous week was spent doing little actual research but a lot of Julia coding.

I could not have finished the program without the support of the faculty and staff in the NC State Statistics department. Alison McCoy is truly the unsung hero of the department. Several courses will forever stick in my mind as pillars of my statistics education: Linear Models with Dr. Len Stefanski, Statistical Computing with Dr. Hua Zhou, and Bayesian Inference with Dr. Alyson Wilson. It is no coincidence that I asked them to be on my committee, and I thank them for saying yes.

I would not have found myself in the PhD program at NC State if it were not for my incredible statistics professors at Winona State University: Dr. Brant Deppa, Dr. Chris Malone, Dr. Tisha Hooks, and Dr. April Kerby. They invest a lot in their students and have created an amazing program.

I would like to thank all of the open source contributors to the Julia language for the positive, welcoming atmosphere as well as for creating the best language around for technical computing. It is a sincere joy to be surrounded by the brilliant people in that community and to be a part of it. Particularly, I'd like to thank Tom Breloff for writing some of the earliest and best parts of OnlineStats while bearing with me as I learned Julia and git.

Finally, I would like to thank my family and friends from whom I've received nothing but support and kind words of encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 Preliminaries

The focus of this dissertation is online optimization applied to statistics. An *online algorithm* is one which accepts input sequentially, in contrast to an *offline algorithm* where data is input all at once. However, not every online algorithm has an offline counterpart (and vice versa). As a trivial example, consider a sample mean of $n-1$ observations:

$$\theta_{\text{offline}}^{(n-1)} = \frac{1}{n-1}\sum_{i=1}^{n-1} x_i. \tag{1.1}$$

If a single new observation x_n is included, an offline calculation revisits all of the previously seen data, whereas an online update uses only the new observation and the

current estimate:

$$\begin{aligned}\theta^{(n)}_{\text{offline}} &= \frac{1}{n}\sum_{i=1}^{n} x_i, \\ \theta^{(n)}_{\text{online}} &= \left(1 - \frac{1}{n}\right)\theta^{(n-1)}_{\text{online}} + \frac{1}{n}x_n.\end{aligned} \tag{1.2}$$

As datasets in the age of big data get larger, so do the demands on traditional (offline) algorithms for fitting statistics and models. Online algorithms can naturally handle several forms of big data, such as files larger than computer memory (out-of-core processing) or observations arriving continuously in a stream (a common case in quantitative finance). However, the advantage of scaling to big data comes with drawbacks. Many model-fitting algorithms are iterative in nature because there does not exist a closed form solution, e.g. Newton's method for logistic regression. In this case it is not possible for an online algorithm to fit a model to data as well as an offline counterpart; the objective function can be approximately minimized at best. The field of research involved with approximate objective function minimization is called *stochastic approximation*. While there has been increased interest in stochastic approximation in the last decade, there are still many open research questions and room for improvement over the state-of-the-art methods.

Contributions

The main contributions of this dissertation are threefold:

1. Several new methods of stochastic approximation that hold favorable properties over current methods.

2. A unified representation of online algorithms for statistics that can be executed in parallel, implemented in the OnlineStats.jl package.

3. A discussion of the difficulties in comparing stochastic approximation algorithms (ignored in the literature) as well as a novel visualization technique for examining robustness of algorithms to the choice of learning rate.

This introductory chapter will cover the core ideas in statistics, optimization, and stochastic approximation theory from which our contributions draw inspiration.

Notation

To fix notation, let $\theta \in \mathbb{R}^d$ represent a parameter vector of interest. For a given iterative procedure, we denote $\theta^{(t)}$ as the estimated value after the t-th iteration. The objective function we wish to minimize is $\ell(\theta) : \mathbb{R}^d \to \mathbb{R}$. Denote the gradient (column-vector of partial derivatives) as $\nabla\ell(\theta) : \mathbb{R}^d \to \mathbb{R}^d$ and the Hessian matrix as $d^2\ell(\theta) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. When a global minimum of $\ell(\theta)$ exists, denote it as θ^*. Let D_ℓ be the domain of a function $\ell(\theta)$. The norm $\|\cdot\|$ is the L_2 norm unless specified otherwise.

1.1.1 Statistical and Optimization Concepts

Types of Convergence

Statistical consistency (convergence to the true population parameter) is an essential condition for an estimator to be valid. There are multiple notions of convergence for random variables. Three of them are discussed in this dissertation and are defined as follows.

Definition 1.1: Almost sure convergence ($X_t \xrightarrow{a.s.} X$)

A random variable X_t converges *almost surely* (*with probability 1*) to X if

$$P\left(\lim_{t\to\infty} X_t = X\right) = 1. \tag{1.3}$$

Definition 1.2: Convergence in probability ($X_t \xrightarrow{p} X$)

A random variable X_t converges *in probability* to X if for every $\epsilon > 0$,

$$\lim_{t\to\infty} P\left(|X_t - X| > \epsilon\right) = 0. \tag{1.4}$$

Definition 1.3: Convergence in distribution ($X_t \xrightarrow{d} X$)

A random variable X_t converges *in distribution* to X if for every bounded continuous function h,

$$\lim_{t\to\infty} \mathbf{E}[h(X_t)] = \mathbf{E}[h(X)]. \tag{1.5}$$

An important method for deriving the asymptotic distribution for a function of a random variable is called the *Delta Method*.

Theorem 1.4: Delta Method

(Casella and Berger, 2002) If a consistent estimator $\hat{\theta}$ converges in probability to the true value θ^* such that it has an asymptotic distribution

$$\sqrt{n}(\hat{\theta} - \theta^*) \xrightarrow{d} N(0, \Sigma), \tag{1.6}$$

the delta method implies that for a differentiable function $f : \mathbb{R}^d \to \mathbb{R}^k$,

$$\sqrt{n}[f(\hat{\theta}) - f(\theta^*)] \xrightarrow{d} N(0, \nabla f(\theta^*)^T \Sigma \nabla f(\theta^*)). \tag{1.7}$$

Convexity

Convexity is a useful characteristic of a function often used in optimization. Below we give a mathematical definition of convexity as well as some properties of convex functions.

Definition 1.5: Convex Function

A function $f : U \to \mathbb{R}$ is called convex if U is a convex set and

$$f[\gamma\theta_1 + (1-\gamma)\theta_2] \leq \gamma f(\theta_1) + (1-\gamma) f(\theta_2) \tag{1.8}$$

for $0 < \gamma < 1$ and for all $\theta_1, \theta_2 \in U$. A function is called *strictly convex* if the above inequality is strict.

Definition 1.6: Strong Convexity

A function $f : U \to \mathbb{R}$ is called M-strongly convex if

$$f(\theta_1) \geq f(\theta_2) + \nabla f(\theta_2)^T(\theta_1 - \theta_2) + \frac{M}{2}\|\theta_1 - \theta_2\|_2^2, \quad \text{for all } \theta_1, \theta_2 \in U. \tag{1.9}$$

Strong convexity implies the existence of a quadratic lower bound.

Proposition 1.7: Properties of Convex Functions

- (Supporting Hyperplane Inequality) A function f is convex if and only if

$$f(\theta_1) \geq f(\theta_2) + \nabla f(\theta_2)^T(\theta_1 - \theta_2), \quad \text{for all } \theta_1, \theta_2 \in D_f. \tag{1.10}$$

- (Second-order condition for convexity) A twice differentiable function f is convex if and only if $d^2 f(\theta)$ is positive semi-definite for all $\theta \in D_f$. f is strictly convex if $d^2 f(\theta)$ is positive definite for all $\theta \in D_f$.

- (Global optima) For a convex function f on D_f:

  1. Any stationary point is a global minimum.
  2. Any local minimum is a global minimum.
  3. The set of global minima is convex.
  4. If f is strictly convex, the global minimum is unique.

.

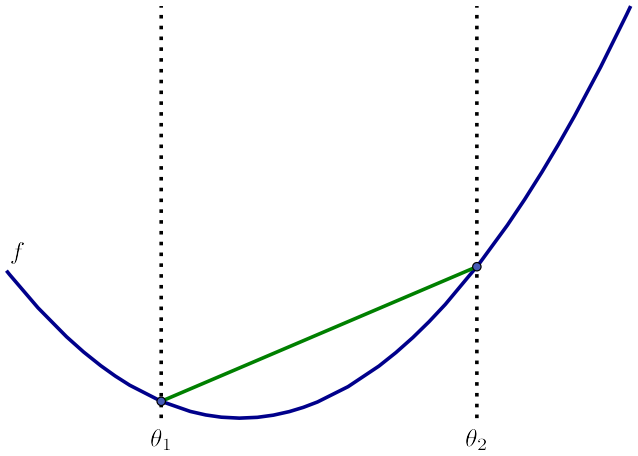


Figure 1.1: Convex Function

The above figure demonstrates a strongly-convex univariate function. Note that the

function lies beneath a straight line connecting any two points on the curve and it has a unique global minimum.

Lipschitz Continuity

A standard regularity condition is *Lipschitz Continuity*.

Definition 1.8: Lipschitz Continuity

A function $f : \mathbb{R}^d \to \mathbb{R}$ is called L-Lipschitz continuous if

$$|f(\theta_1) - f(\theta_2)| \leq L\|\theta_1 - \theta_2\|_2, \quad \text{for all } \theta_1, \theta_2 \in D_f. \tag{1.11}$$

If $L = 1$, f is called a *nonexpansive* mapping. If $L < 1$, f is a *contraction*. When the gradient ∇f is L-Lipschitz continuous, then

$$f(\theta_1) \leq f(\theta_2) + \nabla f(\theta_2)^T(\theta_1 - \theta_2) + \frac{L}{2}\|\theta_1 - \theta_2\|_2^2, \tag{1.12}$$

for all $\theta_1, \theta_2 \in D_f$. This is a common assumption in convergence proofs of optimization algorithms, as it implies there exists a quadratic upper bound of f. Therefore, if a function f is M-strongly convex and has an L-Lipschitz continuous gradient, then f is bounded between two quadratic functions.

1.2 Review of Offline Optimization

Newton's Method

Newton's method is the gold standard in optimization due to its fast (quadratic) convergence. Consider a second-order Taylor expansion around the current iterate $\theta^{(t)}$:

$$\ell(\theta) \approx \ell(\theta^{(t)}) + \nabla\ell(\theta^{(t)})^T(\theta - \theta^{(t)}) + \frac{1}{2}(\theta - \theta^{(t)})^T d^2\ell(\theta^{(t)})(\theta - \theta^{(t)}). \tag{1.13}$$

Setting the gradient of this approximation equal to zero, we get updates of

$$\theta^{(t+1)} = \theta^{(t)} - \left[d^2\ell(\theta^{(t)})\right]^{-1} \nabla\ell(\theta^{(t)}). \tag{1.14}$$

The quadratic convergence and simplicity make Newton's method an attractive candidate for optimization. However, the main drawback is that the calculation and inversion of the Hessian matrix $d^2\ell(\theta^{(t)})$ may be computationally expensive. Another shortcoming is that iterations are not guaranteed to decrease the objective $\ell(\theta)$. To remedy both situations, Newton's method can be generalized to

$$\theta^{(t+1)} = \theta^{(t)} - s_t \left[A^{(t)}\right]^{-1} \nabla\ell(\theta^{(t)}), \tag{1.15}$$

where $A^{(t)}$ is a positive definite approximation of $d^2\ell(\theta^{(t)})$ and s_t is a step length. For sufficiently small s_t, iterates are guaranteed to decrease $\ell(\theta)$. Common choices for the Hessian approximation are $A = I$ (gradient descent) and $A = \mathbf{E}\left[d^2\ell(\theta^{(t)})\right]$ (Fisher scoring method). If a numerical approximation of $d^2\ell(\theta)$ is used rather than an analytical one, this is called a *Quasi-Newton method*.

Expectation - Maximization (EM)

The EM algorithm has many applications in statistics as it provides the machinery for performing maximum likelihood estimation with unobserved or latent data. The latent data statistical model provides a unified framework which covers data with noise, missing data, censored observations, and more (Dempster et al., 1977). EM differs from other algorithms in this review, since it is used to maximize a likelihood in contrast to minimizing a more general objective function.

Let Y be observed data and Z be missing or latent data that makes it easy to maximize the likelihood. Define $g(y, z|\theta)$ as the *complete data likelihood*. The algorithm proceeds by alternating between an expectation step and a maximization step.

- **Expectation Step**

  Evaluate the expectation (with respect to the latent data) of the complete data likelihood, conditioned on the parameter being equal to the current estimate:

$$Q(\theta|\theta^{(t+1)}) = \mathsf{E}_Z\left[\ln g(Y, Z|\theta) \;\middle|\; Y = y, \theta = \theta^{(t)}\right] \tag{1.16}$$

- **Maximization Step**

$$\theta^{(t+1)} = \arg\max_{\theta}\ Q(\theta|\theta^{(t)}) \tag{1.17}$$

The EM algorithm is very stable as each iteration is guaranteed to increase the (observed data) likelihood.

Majorization - Minimization (MM)

The MM algorithm is more of a concept than an actual algorithm. Suppose it is difficult or costly to minimize an objective $\ell(\theta)$ directly. The MM concept is to iteratively create *majorizing* functions and minimize them. MM is extremely stable, as each iteration is guaranteed to decrease the objective. This is similar to the EM algorithm's guaranteed ascent property, due to the fact that EM is a special case of MM. The advantage over EM is that a wider class of problems that can be solved; the objective function does not need to be a likelihood and the majorizing technique does not need to be an expectation.

- **Majorization Step**

  Construct a function $g(\theta|\theta^{(t)})$ such that

$$\begin{array}{rcll} g(\theta|\theta^{(t)}) & \geq & \ell(\theta), & \text{for all } \theta \quad \text{(dominance condition)}, \\ g(\theta^{(t)}|\theta^{(t)}) & = & \ell(\theta^{(t)}) & \text{(tangent condition)}. \end{array} \tag{1.18}$$

- **Minimization Step**

$$\theta^{(t+1)} = \arg\min_{\theta} g(\theta|\theta^{(t)}) \tag{1.19}$$

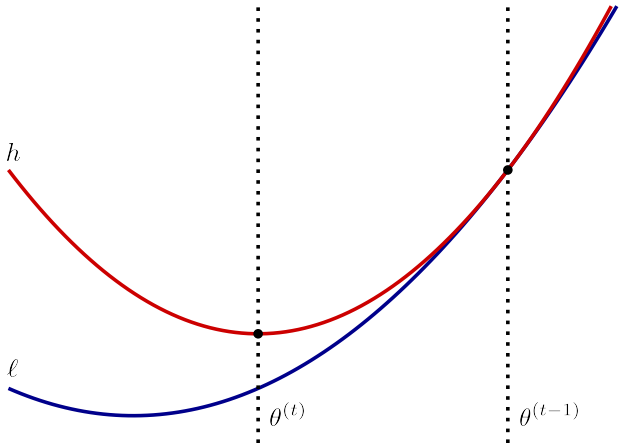


Figure 1.2: Visualization of One MM Algorithm Iteration

The above figure demonstrates a single MM iteration in one dimension. Beginning at $\theta^{(t-1)}$, the majorizing step creates the red line. Then, the minimization step finds the red line's minimizer, which moves $\theta^{(t-1)}$ to $\theta^{(t)}$. Therefore, the MM iteration moves the estimate in the correct direction of the global minimizer.

The key to efficient MM algorithms, constructing majorizations, is a bit of an art form. Hunter and Lange (2004) provide several techniques for creating majorizations:

- **Jensen's Inequality**
  For a convex function ℓ and random variable X, Jensen's inequality states that

$$\ell\left[\mathbf{E}(X)\right] \leq \mathbf{E}\left[\ell(X)\right]. \tag{1.20}$$

  For probability densities $a(x), b(x)$, the fact that $-\ln(x)$ is convex can be used to conclude

$$-\ln\left\{\mathbf{E}\left[\frac{a(X)}{b(X)}\right]\right\} \leq -\mathbf{E}\left\{\ln\left[\frac{a(X)}{b(X)}\right]\right\}. \tag{1.21}$$

  If X has density $b(x)$, the following inequality can be derived:

$$\mathbf{E}\{\ln[a(x)]\} \leq \mathbf{E}\{\ln[b(x)]\}. \tag{1.22}$$

It is this fact that shows the minorizing property of the E-step in the EM algorithm.

- **Minorization via Supporting Hyperplanes**
  If ℓ is convex and differentiable,

$$\ell(\theta) \geq \ell(\theta^{(t)}) + \nabla\ell(\theta^{(t)})^T(\theta - \theta^{(t)}), \tag{1.23}$$

  with equality at $\theta = \theta^{(t)}$. Thus, the right-hand side is a minorization.

- **Majorization via the Definition of Convexity**
  A function ℓ is convex if and only if

$$f\left(\sum_i \alpha_i t_i\right) \leq \sum_i \alpha_i f(t_i) \tag{1.24}$$

  for a finite collection of points t_i and nonnegative multipliers α_i such that $\sum_i \alpha_i = 1$. When ℓ is composed with a linear function $x^T\theta$, substituting $t_i = x_i(\theta_i - \theta_i^{(t)})/\alpha_i + x^T\theta^{(t)}$ creates the inequality (De Pierro, 1995)

$$\ell(x^T\theta) \leq \sum_i \alpha_i \ell\left[\frac{x_i}{\alpha_i}(\theta_i - \theta_i^{(t)}) + x^T\theta^{(t)}\right]. \tag{1.25}$$

- **Majorization via Quadratic Upper Bound**
  Suppose ℓ is twice differentiable and there exists a matrix M such that $M - d^2\ell(\theta)$ is positive semi-definite for all θ. Then there exists a quadratic upper bound

$$\ell(\theta) \leq \ell(\theta^{(t)}) + \nabla\ell(\theta^{(t)})^T(\theta - \theta^{(t)}) + \frac{1}{2}(\theta - \theta^{(t)})^T M(\theta - \theta^{(t)}). \tag{1.26}$$

  Minimizing the right-hand side results in Newton-like updates where the Hessian matrix is replaced with M:

$$\theta^{(t)} = \theta^{(t-1)} - M^{-1}\nabla\ell(\theta^{(t-1)}). \tag{1.27}$$

Coordinate Descent

Coordinate descent algorithms minimize the objective function with respect to one element at a time while keeping the others constant.

$$\theta_j^{(t+1)} = \arg\min_{\theta_j} \ell\left(\theta_1^{(t+1)}, \ldots, \theta_{j-1}^{(t+1)}, \theta_j, \theta_{j+1}^{(t)}, \ldots, \theta_p^{(t)}\right) \tag{1.28}$$

for $j = 1, \ldots, p$.

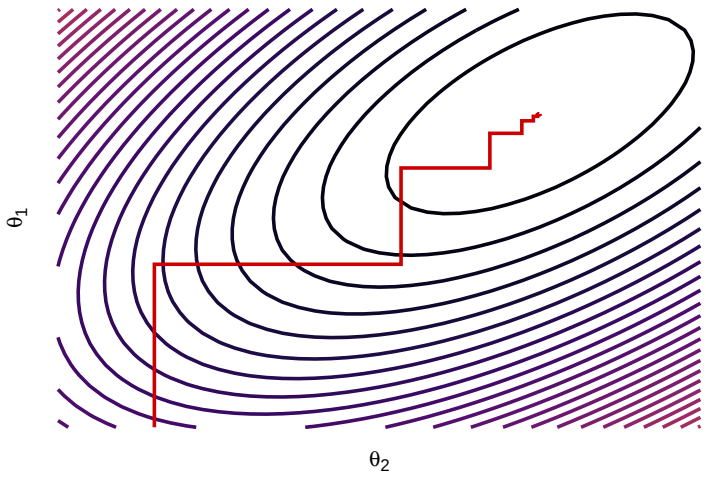


Figure 1.3: Coordinate Descent in Two Dimensions

The figure above demonstrates coordinate descent in two dimensions using a smooth, strongly convex objective. Coordinate descent also has the descent property, but is not guaranteed to converge for a non-differentiable objective. However, it will converge to a global minimum for an objective $\ell(\theta) = f(\theta) + g(\theta)$ where f is convex and differentiable and g is convex. This makes coordinate descent well suited to generalized linear models with regularization (Friedman et al., 2010).

Proximal Gradient Method

The proximal gradient method is a gradient-descent like method with the inclusion of a proximal mapping.

Definition 1.9: Prox Operator

The prox operator or proximal mapping of a convex function g with scaling s is

$$\text{prox}_{sg}(\theta) = \arg\min_u \left(g(u) + \frac{1}{2s}\|u-\theta\|^2 \right). \tag{1.29}$$

Intuitively, the mapping is a compromise between minimizing g and remaining near θ, where the scaling s controls the amount of tradeoff. Also note that the function being minimized by the proximal mapping is a majorizing function of g, so a prox step is an MM update.

Table 1.1: Prox-operator Examples

Function	Prox-operator
$g(\theta) = c$	$\text{prox}_g(\theta_j) = \theta_j$
$g(\theta) = \|\theta\|_1$	$\text{prox}_{sg}(\theta_j) = \text{sign}(\theta_j)\ \max(\|\theta_j\| - s, 0)$
$g(\theta) = \|\theta\|_2$	$\text{prox}_{sg}(\theta) = \theta\ \max\left(1 - \frac{s}{\|\theta\|_2}, 0\right)$
$g(\theta) = \|\theta\|_2^2$	$\text{prox}_{sg}(\theta_j) = \frac{\theta_j}{1+s}$
$g(\theta) = \theta^T A\theta/2 + b^T\theta + c$	$\text{prox}_{sg}(\theta) = (I + sA)^{-1}(\theta - sb)$

The proximal gradient method minimizes a function with two components

$$\ell(\theta) = f(\theta) + g(\theta), \tag{1.30}$$

where f is convex and differentiable and g is a closed convex function with inexpensive prox operator. Iterates take the form

$$\begin{aligned}\theta^{(t+1)} &= \arg\min_\theta \left\{ f(\theta^{(t)}) - \nabla f(\theta^{(t)})^T(\theta + \theta^{(t)}) + \frac{1}{2s_t}\|\theta - \theta^{(t)}\|^2 + g(\theta) \right\} \\ &= \text{prox}_{s_t g}\left(\theta^{(t)} - s_t \nabla f(\theta^{(t)})\right).\end{aligned} \tag{1.31}$$

Thus, the update alternates between a gradient descent step with respect to the differentiable function f and performing a proximal mapping with respect to g. If $s_t < 1/L$ and $f(\theta)$ has an L-Lipschitz continuous gradient, a proximal gradient step is an MM step.

Beck and Teboulle (2009) introduced the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA), an accelerated proximal gradient method. The idea is to perform proximal gradient steps on an extrapolated point.

$$\begin{aligned} y &= \theta^{(t)} + \frac{t-2}{t+1}\left(\theta^{(t)} - \theta^{(t-1)}\right), \\ \theta^{(t+1)} &= \text{prox}_{sg}\left(y - s\nabla f(y)\right). \end{aligned} \tag{1.32}$$

1.3 Review of Online Optimization

Online optimization is almost synonymous with stochastic approximation. Consider the common machine learning objective to minimize an expected loss with respect to an unknown random variable:

$$\arg\min_{\theta} \mathbf{E}_Y[\ell(Y, \theta)]. \tag{1.33}$$

An online algorithm will update parameter estimates $\theta^{(t)}$ based on the previous iterate $\theta^{(t-1)}$ and a random sample $y_t \sim Y$, so the updater has access to all information contained in $\ell(y_t, \theta)$. Some stochastic algorithms are not online, but take random draws of observations from a fixed size dataset. Popular examples of this are the algorithms SAG (Schmidt et al., 2017), SAGA (Defazio et al., 2014), SVRG (Johnson and Zhang, 2013), and MISO (Mairal, 2015). These algorithms follow the form of (1.33) by replacing Y with the empirical distribution of a sample.

1.3.1 Stochastic Approximation

Robbins-Monro Algorithm

The foundational paper of stochastic approximation is Robbins and Monro (1951), which presents a stochastic algorithm for finding the root of an unknown function from which noisy observations can be taken. The interpretation by Lai (2003) is presented

here. Suppose we wish to use successive approximations $\theta^{(t)}$ to find the unique root θ^* of a function $M(\theta)$ based on observations

$$y_t = M(\theta^{(t-1)}) + \epsilon_t, \tag{1.34}$$

where ϵ_t are unobservable random errors with mean 0. The Robbins-Monro algorithm is

$$\begin{aligned} \theta^{(t)} &= \theta^{(t-1)} - \gamma_t y_t, \\ &= \theta^{(t-1)} - \gamma_t \left[M(\theta^{(t-1)}) + \epsilon_t \right] \end{aligned} \tag{1.35}$$

where $\{\gamma_t\}_{t=1}^{\infty}$ is a positive sequence such that $\sum \gamma_t = \infty$, $\sum \gamma_t^2 < \infty$. Convergence to θ^* is achieved under the assumption $M(\theta^{(t)}) > 0$ when $\theta^{(t)} > \theta^*$ and $M(\theta^{(t)}) < 0$ when $\theta^{(t)} < \theta^*$.

Kiefer-Wolfowitz Algorithm

For finding a critical point of a function, the Robbins-Monro algorithm is modified by Kiefer and Wolfowitz (1952) to use updates of the form

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \left[\frac{y_t(\theta^{(t-1)} + c_t) - y_t(\theta^{(t-1)} - c_t)}{2c_t} \right], \tag{1.36}$$

where c_t is a positive sequence such that $\sum \gamma_t c_t < \infty$. In other words, the derivative is estimated with finite differences.

The approach to prove convergence used by Robbins and Monro (1951) and Kiefer and Wolfowitz (1952) starts with showing $\mathbf{E}[(\theta^{(t)} - \theta^*)^2]$ converges to some limit in L^2. Then, a contradiction is found if $\theta^{(t)}$ does not converge to θ^* in probability. Blum (1954) proved almost sure convergence for the Robbins-Monro under assumptions

$$\begin{aligned} |M(\theta^{(t)})| &\leq c(|\theta^{(t)} - \theta^*| + 1) \quad \text{for all } \theta^{(t)} \text{ and some } c > 0, \\ \inf_{\epsilon \leq |\theta^{(t)} - \theta| \leq 1/\epsilon} \left[M(\theta^{(t)})(\theta^{(t)} - \theta^*) \right] &> 0 \quad \text{for all } 0 < \epsilon < 1, \end{aligned} \tag{1.37}$$

and almost sure convergence for Kiefer-Wolfowitz after removing the assumption $\sum \gamma_t c_t < \infty$.

Nonnegative Almost Supermartingale

A concept that appears in convergence proofs of stochastic approximation algorithms is the idea of a *filtration*. A filtration can be interpreted as the formal concept of history or past information.

Definition 1.10: Filtration

Let $(\Omega, \mathcal{F}, P)$ be a probability space with σ-algebra $\mathcal{F}$ and probability measure P. A filtration is an increasing sequence of σ-algebras

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots \subset \mathcal{F}. \tag{1.38}$$

A sequence of random variables X_t always has the natural filtration where $\mathcal{F}_t = \sigma(X_1, \ldots, X_t)$ is the σ-algebra generated by the random sequence up to t. By property of $\mathcal{F}_t$-measurable random variables, $\mathbf{E}(X_t|\mathcal{F}_t) = X_t$. If $\mathbf{E}(|X_t|) < \infty$ for all t and $\mathbf{E}[X_t|\mathcal{F}_s] = X_s$ for all $s < t$, $\{X_t\}$ is said to be *adapted to the filtration* $\{\mathcal{F}_t\}$.

Definition 1.11: Nonnegative Almost Supermartingale

Suppose there exists nonnegative $\mathcal{F}_t$-measurable random variables M_t, A_t, B_t, C_t such that

$$\mathbf{E}(M_{t+1}|\mathcal{F}_t) \leq (1 + A_t)M_t - B_t + C_t. \tag{1.39}$$

If $\sum A_t < \infty$ and $\sum C_t < \infty$ almost surely, then M_t converges to a finite limit and $\sum B_t < \infty$ almost surely. A sequence that satisfies (1.39) is called a *nonnegative almost supermartingale*.

Robbins and Siegmund (1985) proved convergence of the Robbins-Monro algorithm using almost supermartingale theory with $M_t = (\theta^{(t)} - \theta^*)^2$.

Connection with Ordinary Differential Equations

The proof techniques in this dissertation are based on martingales. However, another rather interesting method for proving convergence is based on equating Robbins-

Monro updates with an ordinary differential equation. Let $s_t = \sum_{i=1}^{t} \gamma_i$ be the sum of weights up until point t. The updates can then be interpreted as a function of "time". Define

$$\theta(s) = \frac{\theta^{(t)}(s_{t+1} - s) + \theta^{(t+1)}(s - s_t)}{s_{t+1} - s_t}, \quad s_t \leq s < s_{t+1}. \tag{1.40}$$

That is, $\theta(s_t) = \theta^{(t)}$ and $\theta(s)$ is the linear interpolation between $\theta^{(t)}, \theta^{(t+1)}$ when $s_t < s < s_{t+1}$. Rearranging the Robbins Monro update (1.35) and rewriting estimates as above, we get

$$\begin{aligned} \theta^{(t)} &= \theta^{(t-1)} - \gamma_t[M(\theta^{(t-1)}) + \epsilon_t], \\ \frac{\theta^{(t)} - \theta^{(t-1)}}{\gamma_t} &= -M(\theta^{(t-1)}) - \epsilon_t, \\ \frac{\theta(s_t) - \theta(s_{t-1})}{s_t - s_{t-1}} &= -M(\theta(s_{t-1})) - \epsilon_t. \end{aligned} \tag{1.41}$$

Since $s_t - s_{t-1} = \gamma_t \to 0$, this is strikingly similar to the ordinary differential equation

$$\frac{d\theta}{ds} = -M[\theta(s)]. \tag{1.42}$$

Under regularity conditions, $\theta^{(t)} \to \theta^*$ if θ^* is a global asymptotically stable equilibrium of the ODE. For technical details, we refer the reader to Kushner and Yin (2003).

Stochastic Gradient Descent (SGD)

Stochastic gradient descent was introduced by Sakrison (1965) and is a straightforward application of the Robbins-Monro algorithm. It is fairly intuitive and easy to implement for a wide variety of optimization problems. In recent years, SGD-like algorithms have been popular for online machine learning problems as it is trivially scalable to big data.

Putting SGD in the Robbins-Monro form, we wish to solve $M(\theta) = \mathbf{E}_Y[\nabla \ell(Y, \theta)] = 0$. This expectation cannot be observed, but we can obtain $M(\theta) + \epsilon_t$ where $\epsilon_t = \nabla \ell(y_t, \theta) - M(\theta)$ and y_t are random samples from the distribution of Y. The updates are then

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \nabla \ell(y_t, \theta^{(t-1)}). \tag{1.43}$$

Asymptotic Normality

Chung (1954) and Sacks (1958) were the first to investigate the asymptotic distribution for the Robbins-Monro algorithm. Under the learning rate $\gamma_t = (t\beta)^{-1}$ where $\beta = M'(\theta^*)$, the univariate Robbins-Monro iterates satisfy

$$\sqrt{t}(\theta^{(t)} - \theta^*) \to N\left(0, \frac{\sigma^2}{\beta^2}\right), \tag{1.44}$$

where $\sigma^2 = \lim_{t\to\infty} \mathbf{E}(\epsilon_t|\mathcal{F}_{t-1})$. The extension to the multivariate case is given in Fabian (1968).

More recently, Toulis et al. (2017) derived asymptotic normality for a variety of stochastic gradient updates of the form

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t C_t \nabla \ell_t(\theta^{(t-1)}), \tag{1.45}$$

where $\ell_t(\theta) = \ell(y_t, \theta)$, $\gamma_t > 0$ and C_t is a positive definite matrix.

Assumption 1.12: Assumptions for Normality of Stochastic Gradient Update

(a) The step size sequence is $\gamma_t = \eta/t^r, \eta > 0, r \in (0.5, 1]$.

(b) The objective $\ell(\theta)$ is convex, L-Lipschitz continuous, twice differentiable, and minimized at θ^*.

(c) The Hessian matrix for online objectives have positive trace for all t: trace$[d^2\ell_t(\theta)] > 0$. The eigenvalues of the expected second derivative are bounded: $0 < \lambda_j < \infty$ for all eigenvalues λ_j of $\mathbf{E}[d^2\ell_t(\theta)]$.

(d) Each matrix C_t is a positive definite matrix such that $C_t = C + O(\gamma_t)$ where C is symmetric, positive definite and commutes with $\mathbf{E}[d^2\ell(\theta^*)]$. The eigenvalues of each C_t are bounded: $0 < \lambda_{tj} < \infty$ for all eigenvalues λ_{tj} of C_t.

(e) The Lindeberg conditions for asymptotical normality are satisfied. Let $\Xi_t = \mathbf{E}[\nabla\ell_t(\theta^*)\nabla\ell_t(\theta^*)^T|\mathcal{F}_{t-1}]$. Then $\|\Xi_t - \Xi\| = O(1)$ for all t, and $\|\Xi_t - \Xi\| \to 0$ for a symmetric positive definite matrix Ξ. Let $\sigma^2_{t,s} = \mathbf{E}[1_{\|\xi_t(\theta^*)\|^2 \geq s/\gamma_t}\|\xi_t(\theta^*)\|^2]$. Then for all $s > 0$, $\sum_{i=1}^t \sigma^2_{i,s} = o(n)$ if $\gamma_t = 1$ and $\sigma^2_{t,s} = o(1)$ otherwise.

Theorem 1.13: Asymptotic Normality of Stochastic Gradient Algorithms

Under the above assumptions where $r = 1$ and $2\eta C d^2 \ell(\theta^*) - I$ is positive definite, the updates $\theta^{(t)}$ from (1.45) are asymptotically normal with mean θ^* and variance

$$\frac{\eta^2 [2\eta C d^2 \ell(\theta^*) - I]^{-1} C d^2 \ell(\theta^*) C}{t}. \tag{1.46}$$

Polyak-Ruppert Averaging

While not an algorithm itself, averaging techniques (Polyak and Juditsky, 1992; Ruppert, 1988) can often accelerate convergence of stochastic approximation methods. The motivating idea is that averaging the iterates reduces variability. In practice, it is common to let the algorithm learn for a finite number of samples t_0 and then begin averaging:

$$\bar{\theta}^{(t)} = \frac{1}{t - t_0} \sum_{i=t_0}^{t} \theta^{(i)}. \tag{1.47}$$

Online EM Algorithm

An online version of the EM algorithm in Cappé and Moulines (2009) is based on minimizing a stochastic approximation of the expectation step. To ease the transition to the online case, here we present the (offline) EM algorithm in a slightly different form. Assume the negative loglikelihood is normalized by n, the number of observations:

$$\ell(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(\theta), \tag{1.48}$$

so that $\ell(\theta)$ is majorized by

$$Q_t(\theta) = \frac{1}{n} \sum_{i=1}^{n} g_i(\theta | \theta^{(t-1)}), \tag{1.49}$$

where $g_i(\theta|\theta^{(t-1)})$ majorizes $\ell_i(\theta)$ at $\theta^{(t-1)}$ for all i. The E-step and M-step can therefore

be written as

$$\begin{aligned} Q_t(\theta) &= \frac{1}{n}\sum_{i=1}^{n} g_i(\theta|\theta^{(t-1)}), \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta). \end{aligned} \tag{1.50}$$

For the online EM algorithm, we observe a sequence of independent random samples y_t. For each sample, a majorizing expectation is created such that $g(y_t, \theta|\theta^{(t-1)})$ majorizes $\ell(y_t, \theta)$ at $\theta^{(t-1)}$. The online EM algorithm then performs the updates

$$\begin{aligned} Q_t(\theta) &= (1-\gamma_t)Q_{t-1}(\theta) + \gamma_t g(y_t, \theta|\theta^{(t-1)}), \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta), \end{aligned} \tag{1.51}$$

where $\{\gamma_t\}_{t=1}^{\infty}$ satisfies $\sum \gamma_t = \infty, \sum \gamma_t^2 < \infty$. The proof of consistency in Cappé and Moulines (2009) relies on assuming the complete data likelihood comes from an exponential family. The stochastic approximation (E) step thus amounts to approximating the sufficient statistics from the complete data likelihood.

1.4 Discussion

Many of the offline algorithms in this chapter have stochastic approximation counterparts. Just as Newton's method is the gold standard for offline optimization, stochastic gradient descent (the online counterpart to gradient descent) is the gold standard for online optimization. Most of the recent developments in online optimization are variants of SGD. The main drawback to SGD-like algorithms is that there is information "left on the table" at each update. By only using gradient information, there is no guarantee of reducing the objective function for even the current observation. The first few iterations of stochastic gradient algorithms often do more harm than good, forcing future iterations to "undo damage" before converging to the correct solution. One can ensure stability in finite samples with a careful selection of hyper-parameters (step size sequence $\{\gamma_t\}$), but this adds complexity to already-complex methods. This dissertation proposes using MM concepts to acquire more stability than that which is offered by stochastic gradient algorithms. Intuitively, MM algorithms incorporate some second-order information of the objective into the majorizing function and thus updates are

less sensitive to step sizes. Little has been done to incorporate the MM algorithm into the world of stochastic approximation, but this dissertation shows there is much to be gained by doing so.

The algorithms presented in chapters two and three develop a methodology for using MM concepts in the stochastic approximation setting. Chapter four describes the Julia package OnlineStats, which implements the algorithms defined in the preceding chapters. Chapter five discusses the difficulties in comparing stochastic approximation algorithms, develops a novel framework that alleviates these difficulties, and compares the algorithms from chapters two and three alongside state-of-the-art stochastic gradient algorithms.

CHAPTER

2

ONLINE MM ALGORITHMS

2.1 Introduction

Majorization-minimization (MM) is a common optimization principle with a wide variety of uses in statistics and machine learning. It is an intuitive process that decreases the value of the objective function with each iteration, and many existing algorithms can be interpreted as MM, such as the EM-algorithm (Dempster et al., 1977). Despite its usefulness, it is often difficult to adapt MM algorithms to large-scale problems. This problem is not unique to MM algorithms, as any iterative solver that requires the entire dataset be available faces the same struggles. Offline algorithms such as gradient descent and quasi-Newton methods have online analogues: stochastic gradient descent (Sakrison, 1965), stochastic quasi-Newton (Schraudolph et al., 2007), and second-order stochastic gradient descent (Bottou et al., 2017). However, little has been done to adapt MM algorithms to online optimization.

In the online learning setting, the goal is to find

$$\arg\min_{\theta} \mathbf{E}_Y[\ell(Y, \theta)], \tag{2.1}$$

where the expectation cannot be evaluated, but the learner has access to a sequence of random samples $\{y_t\}_{t=1}^{\infty}$ from Y. A learner updates the estimate $\theta^{(t)}$ from the sample y_t and the previous estimate $\theta^{(t-1)}$ using the information contained in $\ell_t(\theta) = \ell(y_t, \theta)$. Many online learning algorithms are variants of stochastic gradient descent (SGD), which is defined by minimizing a quadratic approximation for a positive step size γ_t:

$$\begin{aligned} \theta^{(t)} &= \arg\min_{\theta} \left\{ \ell_t(\theta^{(t-1)}) + \nabla \ell_t(\theta^{(t-1)})^T(\theta - \theta^{(t-1)}) + \frac{1}{2\gamma_t}\|\theta - \theta^{(t-1)}\|^2 \right\} \\ &= \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)}). \end{aligned} \tag{2.2}$$

Since SGD-like algorithms use only the gradient, all other information contained in $\ell_t(\theta)$ is ignored. The algorithms introduced in this chapter naturally incorporate more information through the use of majorizing functions, resulting in more stable updates.

The Majorization-minimization (MM) principle has only recently been used in online learning. One of the two algorithms presented in this paper is the stochastic MM algorithm from Mairal (2013b) that provides convergence for convex and nonconvex problems under iterate averaging schemes only. We contribute convergence results for stochastic MM (which we call OMAS) without averaging. Just as MM algorithms are a generalization of the EM algorithm, the stochastic MM algorithm is a generalization of the online EM algorithm in Cappé and Moulines (2009). Mairal (2015) and Mairal (2013a) describe an incremental MM scheme called MISO (Minimization by Incremental Surrogate Optimization), similar in spirit to the SAG (Stochastic Average Gradient) algorithm of Schmidt et al. (2017). SAG and MISO are hybrid stochastic/deterministic schemes that are not online algorithms and require the data to be fixed in size. While the MM principle is not explicitly mentioned, Ryu and Boyd (2014) and Toulis and Airoldi (2015) use proximal mappings in online updates, which is a specific kind of MM step.

MM Algorithms

The MM algorithm is more of a concept than it is an actual algorithm. Suppose it is difficult or costly to minimize an objective $\ell(\theta)$ directly. The MM concept is to iteratively create majorizing functions and minimize them. MM is extremely stable, as each iteration is guaranteed to decrease the objective. As a generalization of the EM algorithm, the objective function does not need to be a likelihood and the majorizing technique does not need to be an expectation. An introduction to (offline) MM algorithms is given in Hunter and Lange (2004), with a more thorough treatment in Lange (2016).

- **Majorization Step**

  Construct a surrogate objective function h such that

$$h(\theta|\theta^{(t-1)}) \geq \ell(\theta), \quad \text{with strict equality at } \theta = \theta^{(t-1)}. \tag{2.3}$$

- **Minimization Step**

  Minimize h:

$$\theta^{(t)} = \arg\min_{\theta} h(\theta|\theta^{(t-1)}). \tag{2.4}$$

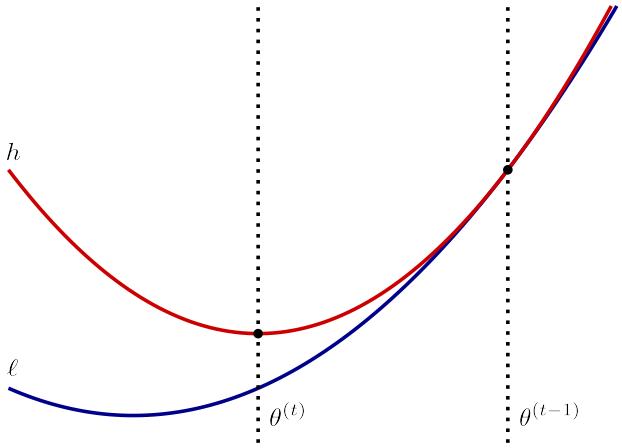


Figure 2.1: Visualization of One MM Algorithm Iteration

The figure above shows the descent property of MM algorithms: By minimizing a surrogate function h, the estimate moves closer to the optimal value of l. The surrogate h incorporates some of the second order information of l and therefore guarantees that the update doesn't "jump too far".

Constructing majorizing functions is a bit of an art form. A common method of construction is via a quadratic upper bound (Hunter and Lange, 2004). Let $\ell(\theta)$ be a twice differentiable objective function and suppose there exists a matrix A such that $A - d^2\ell(\theta)$ is positive semi-definite. Then, the function

$$h(\theta|\theta^{(t)}) = \ell(\theta^{(t)}) + \nabla\ell(\theta^{(t)})^T(\theta - \theta^{(t)}) + \frac{1}{2}(\theta - \theta^{(t)})^T A(\theta - \theta^{(t)}) \tag{2.5}$$

majorizes $\ell(\theta)$ at $\theta^{(t)}$. Minimizing this surrogate creates updates of the form

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t A^{-1}\nabla\ell(\theta^{(t-1)}). \tag{2.6}$$

Therefore, quadratic upper bound MM updates look similar to Newton's method, but use an approximation of the Hessian matrix that guarantees descent. Note that if A is diagonal, updates can be performed element by element, and it is not necessary to solve a linear system at each iteration. Furthermore, if $\ell(\theta)$ has an L-Lipschitz continuous gradient and I is the identity matrix of appropriate size, $LI - d^2\ell(\theta)$ is positive semi-definite and $A = LI$ can be used in the majorization.

The following two propositions are useful in constructing quadratic upper bounds for models that are linear in the parameters, i.e. the objective has the form $\ell_t(\theta) = f_t(x_t^T\theta)$. In this case, the second derivative is $f_t''(x_t^T\theta)x_t x_t^T$. Therefore, if $f_t''(x_t^T\theta)$ is bounded for all θ for some constant c and $A - x_t x_t^T$ is positive semidefinite, then $cA - d^2\ell_t(\theta)$ is positive semidefinite.

Proposition 2.1

Let diag(A) denote the diagonal matrix containing the diagonal elements of A. For a matrix $X = (x_1, x_2, \ldots, x_p) \in \mathbb{R}^{n \times p}$, $p\,\mathsf{diag}(X^T X) - X^T X$ is positive semi-definite.

Proof. Let $a \in \mathbb{R}^p$. Let $X = (x_1, x_2, \ldots, x_p)$. Then

$$
\begin{aligned}
a^T \left[p\,\mathsf{diag}(X^T X) - X^T X \right] a &= p\, a^T \mathsf{diag}(X^T X) a - a^T X^T X a \\
&= p \sum_{j=1}^{p} a_j^2 x_j^T x_j - \sum_{i=1}^{p} \sum_{j=1}^{p} a_i a_j x_i^T x_j \\
&= p \sum_{j=1}^{p} \|v_j\|^2 - \sum_{i=1}^{p} \sum_{j=1}^{p} v_i^T v_j \quad \text{for } v_i = a_i x_i \\
&= (p-1) \sum_{j=1}^{p} \|v_j\|^2 - 2 \sum_{i=1}^{p} \sum_{j>i} v_i^T v_j \\
&= \sum_{i=1}^{p} \sum_{j>i} \left[(\|v_i\| - \|v_j\|)^2 + 2\|v_i\|\|v_j\| \right] - 2 \sum_{i=1}^{p} \sum_{j>i} v_i^T v_j \\
&\geq \sum_{j=1}^{p} \sum_{i>j} 2\|v_i\|\|v_j\| - 2 v_i^T v_j \\
&\geq 0 \quad \text{by Cauchy-Schwarz inequality.}
\end{aligned}
$$

□

Proposition 2.2

For $x \in \mathbb{R}^d$, $x^T x I - x x^T$ is positive semi-definite.

Proof. Let $a \in \mathbb{R}^d$. Then

$$\begin{aligned} a^T(x^T x I - x x^T)a &= a^T x^T x I a - a^T x x^T a \\ &= a^t a x^T x - a^T x x^T a \\ &= \|a\|^2 \|x\|^2 - (a^T x)^2 \\ &\geq 0 \quad \text{by Cauchy-Schwarz inequality.} \end{aligned} \tag{2.7}$$

□

Online EM Algorithm

The (offline) EM algorithm, introduced in Dempster et al. (1977), has a wide variety of uses in statistics, signal processing, and optimization. The EM idea assumes there is some unobserved or missing data Z that if observed would make the maximum likelihood estimate (MLE) easy to calculate. Let Y be the observed data and $f(Y, Z|\theta)$ be the *complete data* probability density function parameterized by θ. As a specific case of MM, the offline EM algorithm alternates between an expectation (minorization) step and a maximization step. Note that when the objective is maximizing an objective function, MM stands for Minorize-Maximize instead of Majorize-Minimize. To keep the EM algorithm consistent with the algorithms in this chapter (in terms of minimization), we alter the EM update to use the negative loglikelihood:

$$\begin{aligned} Q_t(\theta) &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{E}_Z \left[-\ln f(y_i, z_i|\theta) | \theta^{(t-1)} \right], \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta). \end{aligned} \tag{2.8}$$

The precursor to Stochastic MM is the online EM algorithm in Cappé and Moulines (2009), based on a stochastic approximation of the expectation:

$$\begin{aligned} Q_t(\theta) &= (1 - \gamma_t) Q_{t-1}(\theta) + \gamma_t \mathbf{E}_Z[-\ln(y_t, z_t|\theta)|\theta^{(t-1)}], \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta), \end{aligned} \tag{2.9}$$

where $\gamma_t \in (0, 1)$. Therefore, in translating the EM algorithm to an online update, the M-step remains the same, but the expectation step must use an approximation since only a single observation is available.

2.2 Online MM Algorithms

The two presented algorithms will be jointly referred to as *online MM algorithms*, as the updates are based on majorizations and behave similarly in both theory and practice. For each algorithm, the update is defined in terms of step size $\gamma_t \in (0, 1)$ and $h_t(\theta) = h(y_t, \theta|\theta^{(t-1)})$, a known function of the current iterate and new observation that majorizes $\ell_t(\theta) = \ell(y_t, \theta)$ at $\theta^{(t-1)}$.

2.2.1 Online MM - Averaged Surrogate (OMAS)

Mairal (2013b) introduced the following algorithm as *Stochastic MM*, in which the online EM algorithm is extended to MM algorithms by directly replacing the expectation in equation (2.9) with a more general majorization. The OMAS update is then

$$\boxed{\begin{aligned} Q_t(\theta) &= (1-\gamma_t)Q_{t-1}(\theta) + \gamma_t h_t(\theta) \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta). \end{aligned}} \tag{2.10}$$

2.2.2 Online MM - Averaged Parameter (OMAP)

Rather than averaging surrogate functions, OMAP averages the arguments that minimize the majorizations. Therefore, OMAP is only well-defined for majorizations that have a unique minimum:

$$\boxed{\theta^{(t)} = (1-\gamma_t)\theta^{(t-1)} + \gamma_t \arg\min_{\theta} h_t(\theta).} \tag{2.11}$$

2.2.3 Online MM via Quadratic Upper Bound

Recall that majorization via a quadratic upper bound is a common method of constructing surrogates. If $\ell(\theta)$ is twice differentiable and there exists a matrix A such that $A - d^2\ell(\theta)$ is positive semi-definite for all θ, the function

$$\begin{aligned} h(\theta|\theta^{(t-1)}) &= \ell(\theta^{(t-1)}) + \nabla f(\theta^{(t-1)})^T(\theta - \theta^{(t-1)}) + \frac{1}{2}(\theta - \theta^{(t-1)})^T A(\theta - \theta^{(t-1)}) \\ &= \frac{1}{2}\theta^T H\theta + [\nabla f(\theta^{(t-1)}) - H\theta^{(t-1)}]^T\theta + c, \end{aligned} \tag{2.12}$$

where c includes all the terms that do not depend on θ, majorizes $\ell(\theta)$ at $\theta^{(t-1)}$. Fortunately, closed-form updates can be derived for OMAS and OMAP. This makes it straightforward to translate existing offline MM algorithms via quadratic upper bound into online updates. Updates of this form will be referred to as OMAS-Q and OMAP-Q, respectively.

- **OMAS-Q**

$$\begin{aligned} A_t &= (1-\gamma_t)A_{t-1} + \gamma_t H_t, \\ b_t &= (1-\gamma_t)b_{t-1} + \gamma_t[H_t\theta^{(t-1)} - \nabla\ell_t(\theta^{(t-1)})], \\ \theta^{(t)} &= A_t^{-1}b_t. \end{aligned} \tag{2.13}$$

- **OMAP-Q**

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t H_t^{-1}\nabla\ell_t(\theta^{(t-1)}). \tag{2.14}$$

The difference between OMAS and OMAP is clearly seen in the above updates: OMAS consists of updating "sufficient statistics" while OMAP is directly applied to the parameter.

2.2.4 Regularization in OMAS-Q and OMAP-Q

In machine learning, the objectives $\ell_t(\theta)$ often have the form

$$\ell_t(\theta) = f_t(\theta) + \lambda\psi(\theta), \tag{2.15}$$

where $f_t(\theta)$ is a loss function, $\psi(\theta)$ is a regularization function which penalizes the

size of the coefficients θ, and $\lambda \geq 0$ is a tuning parameter that adjusts the amount of regularization.

Note that majorizing $\ell_t(\theta)$ is possible by majorizing the loss component only and leaving the penalty function as-is. Let $h_t(\theta)$ majorize $f_t(\theta)$ at $\theta^{(t-1)}$. Then

$$h_t(\theta) + \lambda\psi(\theta) \tag{2.16}$$

majorizes $\ell_t(\theta)$ at $\theta^{(t-1)}$. Therefore, it is trivial to add a regularization term into a majorization. The second part of MM, minimization, is also straightforward in the case of OMAS-Q and OMAP-Q. First consider OMAP-Q where $\nabla\ell(\theta)$ is L-Lipschitz continuous and the Hessian matrix approximation is LI for the identity matrix I of appropriate size. The update is then

$$\begin{aligned}\theta^{(t)} &= (1-\gamma_t)\theta^{(t-1)} + \gamma_t \arg\min_\theta \left\{ \nabla\ell_t(\theta^{(t-1)})^T\theta + \frac{L}{2}\|\theta - \theta^{(t-1)}\|^2 + \lambda\psi(\theta) \right\} \\ &= (1-\gamma_t)\theta^{(t-1)} + \gamma_t \text{prox}_{\frac{\lambda}{L}\psi}\left[\theta^{(t-1)} - \frac{1}{L}\nabla\ell_t(\theta^{(t-1)})\right],\end{aligned} \tag{2.17}$$

where the *prox operator* or *proximal mapping* is defined as

$$\text{prox}_\psi(u) = \arg\min_\theta \left\{ \psi(\theta) + \frac{1}{2}\|u-\theta\|^2 \right\}. \tag{2.18}$$

Therefore, the second term in (2.17) is using a proximal gradient step (Parikh et al., 2014) applied to the noisy objective $\ell_t(\theta)$. Now consider OMAS-Q with the same majorization:

$$\begin{aligned}Q_t(\theta) &= (1-\gamma_t)Q_{t-1}(\theta) + \gamma_t h_t(\theta), \\ \theta^{(t)} &= \arg\min_\theta \{Q_t(\theta) + \lambda\psi(\theta)\} \\ &= \arg\min_\theta \left\{ L\theta^T\theta/2 - b_t\theta + \lambda\psi(\theta) \right\} \\ &= \arg\min_\theta \left\{ \lambda\psi(\theta) + \frac{L}{2}\left\|\theta - \frac{b_t}{L}\right\|^2 \right\} \\ &= \text{prox}_{\frac{\lambda}{L}\psi}\left(\frac{b_t}{L}\right),\end{aligned} \tag{2.19}$$

where $b_t = (1-\gamma_t)b_{t-1} + \gamma_t L[\theta^{(t-1)} - \nabla f_t(\theta^{(t-1)})]$.

Regularized OMAS-Q has an advantage over regularized OMAP-Q in regard to variable selection. The prox step for certain penalties (such as LASSO) sets small coefficients equal to zero. OMAP-Q coefficients will only be zero if they are zero in every iteration, since the prox step occurs before the averaging occurs. In OMAS-Q, the prox step occurs after averaging and thus coefficients can be set to zero during any iteration.

2.3 Asymptotic Analysis

Most theoretical guarantees for online learning algorithms are based on convex objectives only. In this section, we show online MM algorithms are not only consistent in the convex case, but converge to a critical point for nonconvex objectives. The theory shows online MM algorithms take steps that are correlated with the stochastic gradient, but naturally incorporate second order information from the stochastic objective. As the *stochastic gradient* is used often in the following analysis, let $g_t = \nabla \ell_t(\theta^{(t-1)})$ with elements $g_t = (g_{t1}, \ldots, g_{td})$. Online MM algorithms will be interpreted to take the form $\theta^{(t)} = \theta^{(t-1)} - \gamma_t \delta_t$, where the "direction" vector has elements $\delta_t = (\delta_{t1}, \ldots, \delta_{td})$.

Convergence of online MM algorithms rely on the theory of *nonnegative almost supermartingales*, discussed in Robbins and Siegmund (1985). The main result is presented below as a lemma.

Lemma 2.3: Almost Supermartingale Convergence

Suppose there exists sequences of nonnegative random variables A_t, B_t, and C_t, adapted to filtration $\mathcal{F}_t$, such that

$$\mathbf{E}[M_{t+1}|\mathcal{F}_t] \leq (1 + A_t)M_t - B_t + C_t. \tag{2.20}$$

If $\sum A_t < \infty$ and $\sum C_t < \infty$ almost surely, then M_t converges to a finite limit and $\sum B_t < \infty$ almost surely.

Assumption 2.4: Online MM Algorithm Assumptions

(a) The step size sequence satisfies

$$\begin{aligned} \gamma_1 &= 1, \\ 0 < \gamma_t &< 1 \text{ if } t > 1, \\ \sum \gamma_t &= \infty, \\ \sum \gamma_t^2 &< \infty. \end{aligned} \tag{2.21}$$

(b) For all t, the stochastic majorizing functions h_t have an L-Lipschitz continuous gradient and are M-strongly convex. Necessarily, $M \leq L$. There exists a constant $B > 0$ such that $\|g_t\|^2 \leq B$ for all t.

(c) Each element of the online MM direction vector satisfies $\text{sign}(\delta_{tj}) = \text{sign}(g_{tj})$ and $|\delta_{tj}| \geq c|g_{tj}|$ for some $c > 0$.

(d) The parameter space is a convex open subset $\Theta \subset \mathbb{R}^d$. $\arg\min_\theta h_t(\theta) \in \Theta$ for all t, the data sequence $\{y_t\}_{t=1}^{\infty}$ is independent and identically distributed, and $\ell(y, \theta)$ is well defined for all $(y, \theta) \in \mathcal{Y} \times \Theta$.

In assumption 2.4, The first two parts of (a) ensure that $Q_1(\theta) = h_t(\theta)$ and that iterates remain inside the parameter space. The second two conditions are standard assumptions for stochastic approximation. Assumption (b) puts regularity conditions on the majorizations that are common in practice. Assumption (c) may at first appear to be a strong condition, but it is satisfied for any majorization that splits parameters into a separable sum

$$h_t(\theta) = \sum_{j=1}^{d} h_{tj}(\theta_j), \tag{2.22}$$

where each component h_{tj} is M-strongly convex and has an L-Lipschitz continuous gradient for a sufficiently large L.

An important property of MM algorithms is the guarantee of a lower objective function value after each iteration (descent property). Online algorithms are unable to claim the same property due to the randomness involved with using one observation at a

time, but online MM algorithms do follow a *stochastic descent property*.

Proposition 2.5: Stochastic Descent Property

Online MM algorithms satisfy the stochastic descent property for all t and any step size $\gamma_t \in (0, 1)$:

$$\ell(y_t, \theta^{(t)}) \leq \ell(y_t, \theta^{(t-1)}). \tag{2.23}$$

Proof. For OMAS,

$$\begin{aligned} Q_t(\theta^{(t)}) &\leq Q_t(\theta^{(t-1)}) \\ (1-\gamma_t)Q_{t-1}(\theta^{(t)}) + \gamma_t h_t(\theta^{(t)}) &\leq (1-\gamma_t)Q_{t-1}(\theta^{(t-1)}) + \gamma_t h_t(\theta^{(t-1)}) \\ (1-\gamma_t)Q_{t-1}(\theta^{(t)}) + \gamma_t h_t(\theta^{(t)}) &\leq (1-\gamma_t)Q_{t-1}(\theta^{(t)}) + \gamma_t h_t(\theta^{(t-1)}) \\ h_t(\theta^{(t)}) &\leq h_t(\theta^{(t-1)}) \\ \ell_t(\theta^{(t)}) \leq h_t(\theta^{(t)}) &\leq h_t(\theta^{(t-1)}) = \ell_t(\theta^{(t-1)}). \end{aligned} \tag{2.24}$$

For OMAP, by definition of majorization and convexity,

$$\begin{aligned} \ell_t(\theta^{(t)}) &\leq h_t(\theta^{(t)}) \\ &= h_t[(1-\gamma_t)\theta^{(t-1)} + \gamma_t \operatorname*{arg\,min}_{\theta} h_t(\theta)] \\ &\leq (1-\gamma_t)h_t(\theta^{(t-1)}) + \gamma_t \min h(\theta) \\ &\leq (1-\gamma_t)h_t(\theta^{(t-1)}) + \gamma_t h_t(\theta^{(t-1)}) \\ &= h_t(\theta^{(t-1)}) \\ &= \ell_t(\theta^{(t-1)}). \end{aligned} \tag{2.25}$$

□

The stochastic descent property is rare among online algorithms. Stochastic gradient algorithms eventually achieve guaranteed descent for a decreasing learning rate (step size sequence), but they can only claim it holds for all t with a careful selection of step sizes with respect to a Lipschitz constant. It is often the case that stochastic gradient algorithms "do harm" (increase the objective value) in the first few iterations, forcing future iterations do "undo the damage" later on. Alternatively, online MM algorithms have guaranteed stochastic descent with few assumptions and without the need for a careful selection of learning rate.

Lemma 2.6: Properties of OMAS Surrogates

Under assumption 2.4 (b), $Q_t(\theta)$ is M-strongly convex with an L-Lipschitz continuous gradient for all t. Furthermore,

$$\nabla Q_t(\theta^{(t-1)}) = \gamma_t g_t. \tag{2.26}$$

Proof. The Lipschitz condition and strong convexity are both proven by induction. $Q_1(\theta) = h_1(\theta)$ has an L-Lipschitz continuous gradient. Assume ∇Q_{t-1} is L-Lipschitz continuous. Then $Q_t(\theta)$ has an L-Lipschitz continuous gradient by definition:

$$\begin{aligned}\|\nabla Q_t(\theta_1) - \nabla Q_t(\theta_2)\| &\leq (1-\gamma_t)\|\nabla Q_{t-1}(\theta_1) - \nabla Q_{t-1}(\theta_2)\| + \gamma_t\|\nabla h_t(\theta_1) - \nabla h_t(\theta_2)\| \\ &\leq (1-\gamma_t)L\|\theta_1 - \theta_2\| + \gamma_t L\|\theta_1 - \theta_2\| \\ &= L\|\theta_1 - \theta_2\|,\end{aligned} \tag{2.27}$$

$Q_1(\theta) = h_1(\theta)$ is M-strongly convex. Assume $Q_{t-1}(\theta)$ is M-strongly convex, so by definition $Q_{t-1}(\theta) - \frac{M}{2}\theta^T\theta$ is convex. Then

$$Q_t(\theta) - \frac{M}{2}\theta^T\theta = (1-\gamma_t)\left[Q_{t-1}(\theta) - \frac{M}{2}\theta^T\theta\right] + \gamma_t\left[h_t(\theta) - \frac{M}{2}\theta^T\theta\right] \tag{2.28}$$

is a weighted average of convex functions and $Q_t(\theta)$ is convex by definition. □

The above lemma states the surrogate objective in OMAS shares some of the same properties as the stochastic majorizations. As the following lemma shows, this result makes it possible to analyze OMAS and OMAP under the same framework.

Lemma 2.7: Online MM Update Bounds

Under assumption 2.4, online MM iterates satisfy

$$\frac{\gamma_t^2}{L^2}\|g_t\|^2 \leq \|\theta^{(t-1)} - \theta^{(t)}\|^2 \leq \frac{\gamma_t^2}{M^2}\|g_t\|^2, \tag{2.29}$$

and therefore converge to a finite point almost surely.

Proof. For OMAS, $Q_t(\theta)$ is M-strongly convex and has an L-Lipschitz continuous gradient. Therefore,

$$\begin{aligned} \frac{1}{L^2}\|\nabla Q_t(\theta^{(t-1)})\|^2 &\leq \|\theta^{(t-1)} - \theta^{(t)}\|^2 \leq \frac{1}{M^2}\|\nabla Q_t(\theta^{(t-1)})\|^2, \\ \frac{1}{L^2}\|g_t\|^2 &\leq \|\theta^{(t-1)} - \theta^{(t)}\|^2 \leq \frac{1}{M^2}\|g_t\|^2. \end{aligned} \tag{2.30}$$

For OMAP, $\|\theta^{(t-1)} - \theta^{(t)}\|^2 = \gamma_t^2 \|\arg\min_\theta h_t(\theta) - \theta^{(t-1)}\|^2$. Since $h_t(\theta)$ is M-strongly convex and has an L-Lipschitz continuous gradient,

$$\frac{\gamma_t^2}{L^2}\|g_t\|^2 \leq \gamma_t^2 \|\arg\min_\theta h_t(\theta) - \theta^{(t-1)}\|^2 \leq \frac{\gamma_t^2}{M^2}\|g_t\|^2. \tag{2.31}$$

In both cases, $\lim_{t\to\infty} \|\theta^{(t-1)} - \theta^{(t)}\|^2 = 0$, so it must be that $\theta^{(t)} \to \theta^{(\infty)}$ for some finite value $\theta^{(\infty)}$ almost surely. □

A result of the above lemma is that by writing online MM iterates as

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \delta_t, \tag{2.32}$$

the direction vector δ_t satisfies $L^{-1}\|g_t\| \leq \|\delta_t\| \leq M^{-1}\|g_t\|$. This direction can be explicitly derived for both OMAS and OMAP:

$$\begin{aligned} \delta_t^{OMAS} &= \frac{\theta^{(t-1)} - \arg\min_\theta Q_t(\theta)}{\gamma_t}, \\ \delta_t^{OMAP} &= \theta^{(t-1)} - \arg\min_\theta h_t(\theta). \end{aligned} \tag{2.33}$$

Lemma 2.8: Online MM Direction and Stochastic Gradient

The update direction is positively correlated with $g_t = \nabla \ell_t(\theta^{(t-1)})$ for all t. Specifically,

$$\frac{\gamma_t}{L}\|g_t\|^2 \leq g_t^T(\theta^{(t-1)} - \theta^{(t)}) \leq \frac{\gamma_t}{M}\|g_t\|^2. \tag{2.34}$$

In other words,

$$\frac{1}{L}\|g_t\|^2 \leq g_t^T \delta_t \leq \frac{1}{M}\|g_t\|^2. \tag{2.35}$$

Proof. For OMAS, $Q_t(\theta)$ is convex and has an L-Lipschitz gradient, which implies co-coercivity of the gradient:

$$\begin{aligned} \left[\nabla Q_t(\theta^{(t-1)}) - \nabla Q_t(\theta^{(t)})\right]^T (\theta^{(t-1)} - \theta^{(t)}) &\geq \frac{1}{L}\|\nabla Q_t(\theta^{(t-1)}) - \nabla Q_t(\theta^{(t)})\|^2 \\ \gamma_t g_t^T(\theta^{(t-1)} - \theta^{(t)}) &\geq \frac{\gamma_t^2}{L}\|g_t\|^2. \end{aligned} \tag{2.36}$$

Similarly for OMAP:

$$\begin{aligned} g_t^T(\theta^{(t-1)} - \theta^{(t)}) &= \gamma_t g_t^T[\theta^{(t-1)} - \arg\min_\theta h_t(\theta)] \\ &\geq \frac{\gamma_t}{L}\|g_t - \nabla h_t[\arg\min_\theta h_t(\theta)]\|^2 \\ &\geq \frac{\gamma_t}{L}\|g_t\|^2. \end{aligned} \tag{2.37}$$

For the upper bound, the Cauchy-Schwarz inequality and lemma 2.7 imply

$$g_t^T(\theta^{(t-1)} - \theta^{(t)}) \leq \|g_t\|\|\theta^{(t-1)} - \theta^{(t)}\| \quad \leq \frac{\gamma_t}{M}\|g_t\|^2. \tag{2.38}$$

□

Lemma 2.9

For some constant $c > 0$, for all t,

$$\nabla \ell(\theta^{(t-1)})^T \mathbf{E}(\delta_t | \mathcal{F}_{t-1}) \geq c \|\nabla \ell(\theta^{(t-1)})\|^2. \tag{2.39}$$

Proof. Directly from assumption 2.4 (c),

$$\begin{aligned} \nabla \ell(\theta^{(t-1)})^T \mathbf{E}(\delta_t | \mathcal{F}_{t-1}) &\geq c \nabla \ell(\theta^{(t-1)})^T \mathbf{E}(g_t | \mathcal{F}_{t-1}) \\ &\geq c \|\nabla \ell(\theta^{(t-1)})\|^2. \end{aligned} \tag{2.40}$$

□

The above lemmas show online MM updates move the estimate in the direction of the stochastic gradient and iterates converge to something. The following convergence theorem shows that the value the iterates converge to is in some sense optimal.

2.3.1 Convergence of Online MM Algorithms

Theorem 2.10: Convergence for Nonconvex Objective

Let $\ell(\theta)$ be differentiable, bounded below, and have an R-Lipschitz continuous gradient. Under assumption 2.4, online MM iterates converge to a stationary point of $\ell(\theta)$ almost surely.

Proof. Let $S_t = \ell(\theta^{(t)})$. Without loss of generality, assume $\ell(\theta) \geq 0$ for all θ. By the quadratic upper bound via the Lipschitz continuous gradient of ℓ,

$$\begin{aligned} S_t &\leq S_{t-1} + \nabla \ell(\theta^{(t-1)})^T(\theta^{(t)} - \theta^{(t-1)}) + \frac{R}{2}\|\theta^{(t)} - \theta^{(t-1)}\|^2 \\ &\leq S_{t-1} - \gamma_t \nabla \ell(\theta^{(t-1)})^T \delta_t + \frac{\gamma_t^2 RB}{2M^2}. \end{aligned} \tag{2.41}$$

Taking expectation with respect to $\mathcal{F}_{t-1}$,

$$\begin{aligned} \mathbf{E}(S_t|\mathcal{F}_{t-1}) &\leq S_{t-1} - \gamma_t \nabla \ell(\theta^{(t-1)})^T \mathbf{E}(\delta_t|\mathcal{F}_{t-1}) + \frac{\gamma_t^2 RB}{2M^2}, \\ &\leq S_{t-1} - \gamma_t c\|\nabla \ell(\theta^{(t-1)})\|^2 + \frac{\gamma_t^2 RB}{2M^2} \quad \text{(by Lemma 2.9)}. \end{aligned} \tag{2.42}$$

Then $S_t \geq 0$, the last term is summable (since $\sum \gamma_t^2 < \infty$), and the middle term is negative. S_t is by definition a nonnegative almost supermartingale, implying

$$c \sum_{t=1}^{\infty} \gamma_t \|\nabla \ell(\theta^{(t-1)})\|^2 < \infty. \tag{2.43}$$

Since $\sum \gamma_t = \infty$, it must be that

$$\begin{aligned} \lim_{t\to\infty} \|\nabla \ell(\theta^{(t-1)})\|^2 &= \|\lim_{t\to\infty} \nabla \ell(\theta^{(t-1)})\|^2 \\ &= \|\nabla \ell(\theta^{(\infty)})\|^2 \\ &= 0. \end{aligned} \tag{2.44}$$

□

2.4 Nonasymptotic Analysis

2.4.1 Stability

One of the main reasons to choose online MM algorithms over stochastic gradient methods is the lack of stability in stochastic gradient descent (SGD), as the lack of the descent property in SGD is often not fixed in its modern variants. Bach and Moulines (2011) derive the following error bounds on SGD iterates, presented here as a theorem.

Theorem 2.11: SGD Error Bound

Assume $\ell_t(\theta)$ is μ-strongly convex with unique minimizer θ^*, $\nabla\ell(\theta)$ is R-Lipschitz continuous for all t, and $\|\nabla\ell_t(\theta^{(t-1)})\|^2 \leq B$ for all t. Define SGD updates as

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \nabla\ell_t(\theta^{(t-1)}), \quad \gamma_t = 1/t^r, \quad r \in (.5, 1). \tag{2.45}$$

Let $M_t = \mathbf{E}(\|\theta^{(t)} - \theta^*\|^2)$. Then,

$$\mathbf{E}(M_t|\mathcal{F}_{t-1}) \leq (1 - 2\mu/t^r + 2R^2/t^{2r})M_{t-1} + \frac{2B}{t^{2r}}. \tag{2.46}$$

By necessity, $R \geq \mu$. It is then common for $(1 - 2\mu/t^r + 2R/t^{2r}) \geq 1$ for a finite number of terms until t gets large enough. This is asymptotically negligible, but the bound can increase dramatically in the first few operations, resulting in poor finite-sample performance. Modern advancements to SGD that incorporate adaptive learning rates, such as ADAGRAD (Duchi et al., 2011), ADAM (Kingma and Ba, 2014), and RMSProp (Tieleman and Hinton, 2012), do not always correct the temporarily growing error bound.

The figure below demonstrates the "contraction term" $(1-2\mu/t^r+2R^2/t^{2r})$ from the above bound for a specific choice of constants. While it only takes eight iterations for the bound to start shrinking (term less than 1), the first seven iterations increase the bound to greater than $63M_0$. In contrast, online MM algorithms are extremely stable in the first few iterations because of the stochastic descent property.

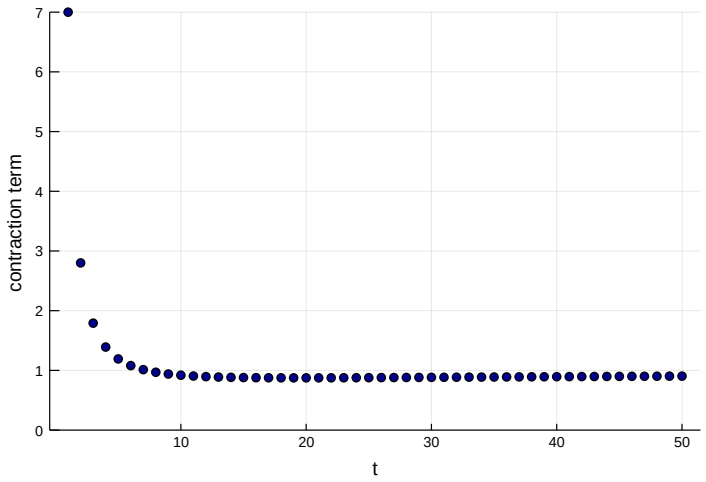


Figure 2.2: "Contraction" term $(1 - 2\mu/t^r + 2R^2/t^{2r})$ of SGD error bound with $r = .7, R = 2, \mu = 1$.

2.4.2 Finite Sample Bounds

It is difficult to apply error bounds to online MM algorithms in general, as the class of functions that could be majorizations is very broad. Here we present finite sample bounds for a specific form of online MM, OMAP-Q, under a fixed step size.

Theorem 2.12: Finite Sample Bounds for OMAP-Q, Fixed Step Size

Assume $\ell(\theta)$ is differentiable with unique minimizer θ^* and each $\ell_t(\theta)$ is μ-strongly convex. Recall the OMAP-Q update is

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t H_t^{-1} g_t, \tag{2.47}$$

where H_t is a matrix such that $H_t - d^2\ell_t(\theta)$ is positive semi-definite for all θ. Assume $\nabla\ell_t(\theta)$ is L_t-Lipschitz continuous and set $H_t = L_t I$. Define $\underline{L} = \inf L_t, \bar{L} = \sup L_t$. Necessarily, $\mu < \underline{L} \leq \bar{L}$. Then under assumption 2.4 and using constant step size $\gamma < 1/2$,

$$\mathbf{E}(\|\theta^{(t)} - \theta^*\|^2 | \mathcal{F}_{t-1}) \leq \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^t \|\theta^{(0)} - \theta^*\|^2 + \frac{\gamma\bar{L}}{2\underline{L}^2\mu} B. \tag{2.48}$$

Proof.

$$\begin{aligned} S_t &= \|\theta^{(t)} - \theta^*\|^2 \\ &= \|\theta^{(t-1)} - \theta^* - \gamma_t L_t^{-1} g_t\|^2 \\ &= S_{t-1} + \frac{\gamma_t^2}{L_t^2}\|g_t\|^2 - 2\frac{\gamma_t}{L_t} g_t^T(\theta^{(t-1)} - \theta^*). \end{aligned} \tag{2.49}$$

Taking expectation,

$$\begin{aligned} \mathbf{E}(S_t|\mathcal{F}_{t-1}) &\leq S_{t-1} + \frac{\gamma_t^2}{L_t^2} B - 2\frac{\gamma_t}{L_t}\nabla\ell(\theta^{(t-1)})^T(\theta^{(t-1)} - \theta^*) \\ &\leq S_{t-1} + \frac{\gamma_t^2}{L_t^2} B - 2\frac{\gamma_t}{L_t}\mu S_{t-1} \quad \text{(by strong convexity)} \\ &= \left(1 - 2\frac{\gamma_t\mu}{L_t}\right) S_{t-1} + \frac{\gamma_t^2}{L_t^2} B. \end{aligned} \tag{2.50}$$

For a constant step size $\gamma_t = \gamma$,

$$\begin{aligned} \mathbf{E}(S_t|\mathcal{F}_{t-1}) &\leq \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^t S_0 + \frac{\gamma^2}{\underline{L}^2} B \sum_{i=0}^{t-1} \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^i \\ &\leq \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^t S_0 + \frac{\gamma^2}{\underline{L}^2} B \sum_{i=0}^{\infty} \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^i \\ &= \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^t S_0 + \frac{\gamma^2}{\underline{L}^2} B \frac{L}{2\gamma\mu} \\ &= \left(1 - 2\frac{\gamma\mu}{\bar{L}}\right)^t S_0 + \frac{\gamma\bar{L}}{2\underline{L}^2\mu} B. \end{aligned} \tag{2.51}$$

□

2.5 Examples

In this section, we derive online MM algorithms for a variety of statistical learning problems. The behavior of the following algorithms and comparison to state-of-the-art methods is provided in chapter 5.

2.5.1 Linear Regression

The loss function (negative log-likelihood) for a single observation in a linear regression model is

$$f_t(\beta) = \frac{1}{2}(y_t - x_t^T\beta)^2, \quad y_t \in \mathbb{R}, x_t \in \mathbb{R}^d. \tag{2.52}$$

The second derivative is $d^2 f_t(\beta) = x_t x_t^T$. By proposition 2.2, OMAS-Q and OMAP-Q can use the Hessian matrix approximation $x_t^T x_t I$ for an appropriately sized identity matrix I.

OMAS-Q

$$\begin{aligned} a^{(t)} &= (1-\gamma_t)a^{(t-1)} + \gamma_t x_t^T x_t \\ b^{(t)} &= (1-\gamma_t)b^{(t-1)} + \gamma_t[x_t^T x_t \beta^{(t-1)} + (y_t - x_t^T\beta^{(t-1)})x_t] \\ \beta^{(t)} &= \frac{1}{a^{(t)}} b^{(t)}. \end{aligned} \tag{2.53}$$

OMAP-Q

$$\beta^{(t)} = \beta^{(t-1)} + \frac{\gamma_t}{x_t^T x_t}(y_t - x_t^T\beta^{(t-1)})x_t. \tag{2.54}$$

2.5.2 Logistic Regression

The loss function (negative log-likelihood) for a single observation in a linear regression model is

$$f_t(\beta) = \ln(1 + e^{y_t x_t^T \beta}), \quad y_t \in [-1, 1], x_t \in \mathbb{R}^d. \tag{2.55}$$

The second derivative is

$$d^2 f_t(\beta) = \hat{p}_t(1-\hat{p}_t)x_t x_t^T, \tag{2.56}$$

where $\hat{p}_t$ is the predicted probability that $y_t = 1$. Since $\hat{p}_t \in [0,1]$, $0 \leq \hat{p}_t(1-\hat{p}_t) \leq 1/4$. Just as for linear regression, OMAS-Q and OMAP-Q can use proposition 2.2 to derive the Hessian matrix approximation $A_t = \frac{1}{4}x_t^T x_t I$.

OMAS-Q

$$\begin{aligned} a^{(t)} &= (1-\gamma_t)a^{(t-1)} + .25\gamma_t x_t^T x_t \\ b^{(t)} &= (1-\gamma_t)b^{(t-1)} + \gamma_t \left[.25 x_t^T x_t \beta^{(t-1)} + \frac{y_t}{1+e^{-y_t x_t^T \beta^{(t-1)}}} x_t \right] \\ \beta^{(t)} &= \frac{1}{a^{(t)}} b^{(t)}. \end{aligned} \tag{2.57}$$

OMAP-Q

$$\beta^{(t)} = \beta^{(t-1)} + \gamma_t \left(\frac{.25}{x_t^T x_t} \right) \frac{y_t}{1+e^{-y_t x_t^T \beta^{(t-1)}}} x_t. \tag{2.58}$$

2.5.3 Generalized Distance Weighted Discrimination (DWD)

An extension of support vector machine's (SVM) $L1$-Hinge Loss, the generalized distance weighted discrimination loss is parameterized by a constant q and converges to SVM as $q \to \infty$. The loss and its derivative are:

$$\begin{aligned} v_q(u) &= \begin{cases} 1-u & \text{if } u \leq \frac{q}{q+1} \\ \frac{q^q}{u^q(q+1)^{q+1}} & \text{if } u > \frac{q}{q+1}, \end{cases} \\ v_q'(u) &= \begin{cases} -1 & \text{if } u \leq \frac{q}{q+1} \\ \left[\frac{q}{u(q+1)} \right]^{q+1} & \text{if } u > \frac{q}{q+1}. \end{cases} \end{aligned} \tag{2.59}$$

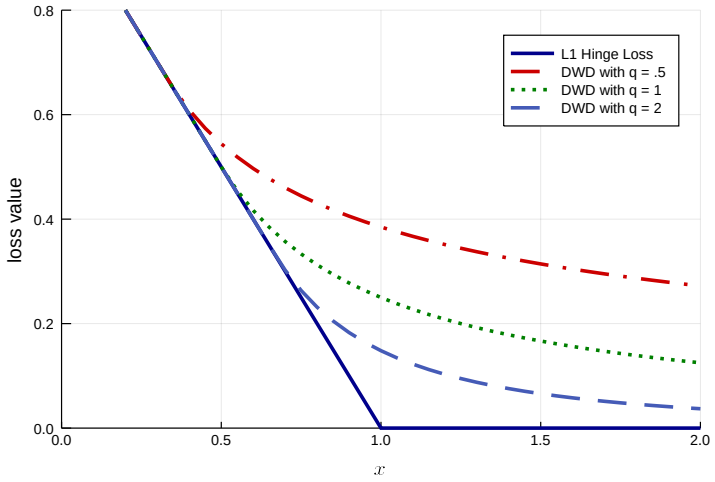


Figure 2.3: DWD Losses compared with Hinge Loss (SVM)

First we present an offline MM algorithm, originally described in Wang and Zou (2015). Let $y \in [-1,1]^n$ be a response vector and let $X = (x_1, \ldots, x_n)^T \in \mathbb{R}^{n \times p}$ be a matrix of predictor variables with observations in rows. The DWD loss is often penalized with an $L2$-penalty and nonnegative tuning parameters $\lambda_j, j = 1, \ldots, p$, providing an objective of the form

$$\ell(\beta) = f(\beta) + \psi(\beta), \tag{2.60}$$

where $f(\beta) = \frac{1}{n}\sum_{i=1}^{n} v_q(y_i x_i^T \beta)$ and $\psi(\beta) = \frac{1}{2}\sum_{j=1}^{p} \lambda_j \beta_j^2$.

The offline MM algorithm in Wang and Zou (2015) relies on a quadratic upper bound and is easily translated into online versions. First, we extend the offline MM algorithm to incorporate elementwise regularization parameters. For each $q > 0$, v_q has a Lipschitz continuous gradient with constant $m_q = (q+1)^2/q$. Thus, the loss in equation (2.59) is majorized at $u^{(t)}$ by

$$g_q(u) = v_q(u^{(t)}) + v_q'(\theta^{(t)})(u - u^{(t)}) + \frac{m_q}{2}(u - u^{(t)})^2. \tag{2.61}$$

Thus, the objective function $\ell(\beta)$ in (2.60) is majorized at $\beta^{(t)}$ by the quadratic:

$$\begin{aligned} g(\beta|\beta^{(t)}) &= f(\beta^{(t)}) + \nabla f(\beta^{(t)})^T(\beta - \beta^{(t)}) + \frac{m_q}{2n}\left\|y^T X(\beta - \beta^{(t)})\right\|_2^2 + \frac{1}{2}\beta^T\Lambda\beta \\ &= f(\beta^{(t)}) + \nabla f(\beta^{(t)})^T(\beta - \beta^{(t)}) + \frac{m_q}{2n}\left\|X(\beta - \beta^{(t)})\right\|_2^2 + \frac{1}{2}\beta^T\Lambda\beta, \end{aligned} \tag{2.62}$$

where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_p)$. Going from the first line to the second in (2.62) is possible because the vector y consists of only positive and negative 1. Taking the derivative of (2.62) respect to β returns

$$\nabla g(\beta|\beta^{(t)}) = \nabla f(\beta^{(t)}) + \frac{m_q}{n}X^T X(\beta - \beta^{(t)}) + \Lambda\beta, \tag{2.63}$$

which by setting equal to zero returns the update rule:

$$\beta^{(t)} = \beta^{(t-1)} - \left(\frac{m_q X^T X}{n} + \Lambda\right)^{-1}\left[\nabla f(\beta^{(t)}) + \Lambda\beta^{(t)}\right] \tag{2.64}$$

Unfortunately, the constant m_q increases quadratically with q and thus the majorization gets "worse" as q increases. As a result, the MM algorithm has slower convergence as the objective function becomes more similar to an $L1$-Hinge loss.

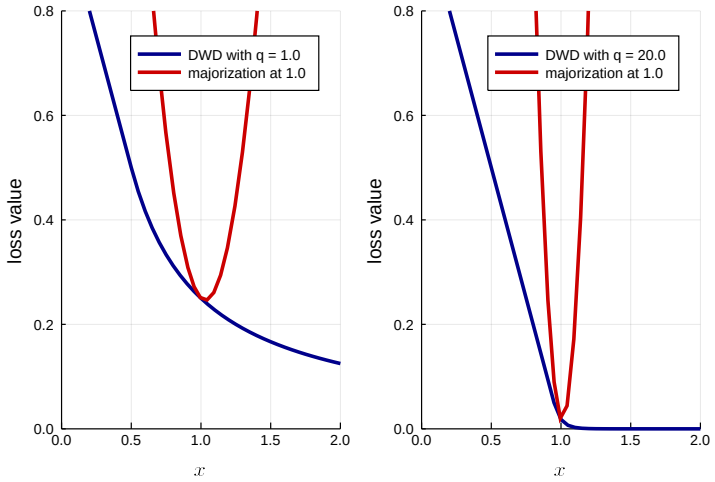


Figure 2.4: Quadratic Upper Bound for DWD Loss

OMAS-Q

$$\begin{aligned} Q_t(\beta) &= \frac{1}{2}\beta^T(A_t + \Lambda)\beta + b_t^T\beta + c_t, \\ \beta^{(t)} &= (A_t + \Lambda)^{-1}b_t, \end{aligned} \tag{2.65}$$

where

$$\begin{aligned} A_t &= (1-\gamma_t)A_{t-1} + \gamma_t m_q x_t^T x_t I, \\ b_t &= (1-\gamma_t)b_{t-1} + \gamma_t[m_q x_t^T x_t \beta^{(t-1)} - v'(y_t x_t^T \beta^{(t-1)})y_t x_t]. \end{aligned} \tag{2.66}$$

OMAS-Q

$$\beta^{(t)} = \beta^{(t-1)} - \frac{\gamma_t}{m_q}(x_t^T x_t I + \Lambda)^{-1}[v_q'(y_t x_t^T \beta^{(t-1)})y_t x_t + \Lambda\beta^{(t-1)}]. \tag{2.67}$$

2.5.4 Dirichlet-Multinomial MLE

Some offline algorithms can be updated in an online fashion. In these cases, it makes more sense to analytically update the sufficient values than to stochastically approximate them. This is similar in spirit to OMAS, but uses an exact analytical update. Consider the probability density function of a d-category Dirichlet-Multinomial distribution with parameter $\alpha = (\alpha_1, \ldots, \alpha_d)$ taking values $x = (x_1, \ldots x_d)$:

$$\begin{aligned} f(x|\alpha) &= \binom{m}{x} \frac{\Gamma(\sum_j \alpha_j)}{\Gamma(m + \sum_j \alpha_j)} \prod_{j=1}^{d} \frac{\Gamma(\alpha_j + x_j)}{\Gamma(\alpha_j)} \\ &= \binom{m}{x} \frac{\prod_{j=1}^{d}(a_j)_{x_j}}{|\alpha|_m}, \end{aligned} \tag{2.68}$$

where $a_y = a(a+1)\ldots(a+y-1)$ denotes the rising factorial, $|\alpha| = \sum \alpha_j$ and $m = |x|$. The loglikelihood for n observations with observation weights w_i is then

$$\ell(\alpha) = -\sum_{k=1}^{m^*-1} r_k \ln(\sum \alpha_j + k) + \sum_{j=1}^{d}\sum_{k=0}^{x^*-1} s_{jk} \ln(\alpha_j + k) + \sum_{i=1}^{n} \ln \binom{mi}{x_i}, \tag{2.69}$$

where $r_k = \sum_i w_i 1_{\{m_i > k\}}$, $s_{jk} = \sum_i w_i 1_{\{x_{ij} > k\}}$, $m^* = \max(m_1, \ldots, m_n)$, $x^* = \max(x_{ij})$. By majorizing the first term via supporting hyperplane inequality and the second term

by Jensen's inequality, a majorizing function for $\ell(\alpha)$ is

$$h(\alpha|\alpha^{(t)}) = \frac{1}{n}\left\{-\sum_{k=0}^{m^*}\frac{r_k}{|\alpha_j^{(t)}|+k}|\alpha| + \sum_{j=1}^{d}\sum_{k=0}^{x^*-1}\frac{s_{jk}\alpha_j^{(t)}}{\alpha_j^{(t)}+k}\ln\alpha_j + c_t\right\}, \tag{2.70}$$

where c_t contains all terms independent of the parameter α. Note that r_k and s_{jk} can be updated analytically. Updates to the parameter then take the form

$$\alpha_j^{(t)} = \left(\sum_{k=0}^{x^*-1}\frac{s_{jk}\alpha_j^{(t-1)}}{\alpha_j^{(t-1)}+k}\right) \Big/ \left(\sum_{k=0}^{m^*-1}\frac{r_k}{|\alpha^{(t-1)}|+k}\right). \tag{2.71}$$

However, if we want online updates to be exactly the same as the offline MM algorithm with t observations, an update to the sufficient values must be followed by several of the iterations in (2.71).

2.5.5 Quantile Regression

Quantile regression is a technique which models the conditional quantile of a distribution, rather than (as with linear regression) the conditional expectation. The offline MM approach in Hunter and Lange (2000) is to majorize a function that closely approximates the quantile loss function $\rho_\tau(u) = u(\tau - 1_{\{u<0\}})$. Let $\rho_\tau^\epsilon(u) = \rho_\tau(u) + \frac{\epsilon}{2}\log(\epsilon+|u|)$. The perturbed objective function is to avoid the possibility of dividing by zero in the majorizer. For a residual $r_i^{(t)} = y_i - x_i^T\beta^{(t)}$, ρ_τ^ϵ is majorized at $\pm r_i^{(t)}$ by

$$h(r|r_i^{(t)}) = \frac{1}{4}\left\{\frac{r^2}{\epsilon+|r_i^{(t)}|} + (4\tau-2)r + c\right\}. \tag{2.72}$$

For $\ell^\epsilon(\beta) = \frac{1}{n}\sum_{i=1}^n \rho_\tau^\epsilon(y_i - x_i^T\beta)$, the quadratic majorizing function is

$$\begin{aligned} Q^\epsilon(\beta|\beta^{(t)}) &= \frac{1}{4n}\sum_{i=1}^{n}\left\{\frac{(y_i - x_i^T\beta)^2}{\epsilon+|r_i^{(t)}|} + (4\tau-2)(y_i - x_i^T\beta) + c_i\right\} \\ &= \frac{1}{4n}\left[(y-X\beta)^T W(y-X\beta) + (4\tau-2)1^T(y-X\beta)\right] + c_t \\ &\propto \frac{1}{2}\beta^T X^T W X\beta - [y^T WX + (2\tau-1)1^T X]\beta + c_t, \end{aligned} \tag{2.73}$$

where $W = \text{diag}[(\epsilon + |r_1^{(t)}|)^{-1}, \ldots, (\epsilon + |r_n^{(t)}|)^{-1}]$ and c_t is a constant containing terms that do not depend on β. The gradient is

$$\nabla Q^{\epsilon}(\beta|\beta^{(t)}) \propto X^T W X\beta - X^T W y - (2\tau - 1)X^T 1, \tag{2.74}$$

and the update is therefore

$$\beta^{(t+1)} = (X^T W X)^{-1} \left[X^T W y + (2\tau - 1)X^T 1\right]. \tag{2.75}$$

In translating the above update to the online setting, we write the single-observation majorizing function as

$$h_t(\beta) = \left\{ \frac{1}{2}\beta^T \frac{x_t x_t^T}{2w_t}\beta - (y_t/(2w_t) + \tau - .5)x_t^T\beta + c_t \right\}, \tag{2.76}$$

where $w_t = \epsilon + |y_t - x_t^T\beta^{(t-1)}|$. The OMAS update is then derived as

$$\begin{aligned} A_t &= (1 - \gamma_t)A_{t-1} + \gamma_t \frac{x_t x_t^T}{2w_t}, \\ b_t &= (1 - \gamma_t)b_{t-1} + \gamma_t \left(\frac{y_t}{2w_t} + \tau - .5\right) x_t, \\ \beta^{(t)} &= A_t^{-1} b_t. \end{aligned} \tag{2.77}$$

Since the majorizing function does not have a unique minimum, OMAP cannot be used.

2.6 Conclusion

In this chapter, we presented two algorithms under the heading *online MM algorithms*. While different, they are based on the same conceptual idea of majorizing functions. OMAS and OMAP are trivially scalable to handle big data, produce favorable finite sample error bounds, and are stable due to the stochastic descent property. Among the desirable properties of online MM algorithms is the fact that it is often straightforward to adapt existing MM algorithms into an online counterpart, specifically in the case of quadratic majorization.

CHAPTER

3

MAJORIZED STOCHASTIC PROXIMAL ITERATION

3.1 Introduction

The proximal mapping is a popular tool in optimization and machine learning. Just as Newton's method is the gold standard for smooth unconstrained problems, proximal algorithms can be viewed similarly for non-smooth constrained problems (Parikh et al., 2014). For a convex function $\ell : \mathbb{R}^d \to \mathbb{R}$ and positive step size γ, the *proximal mapping* (or *prox operator*) is defined as

$$\text{prox}_{\gamma\ell}(\theta^{(t)}) = \arg\min_{\theta} \left\{ \ell(\theta) + \frac{1}{2\gamma}\|\theta - \theta^{(t)}\|_2^2 \right\}. \tag{3.1}$$

The underlying intuition is that the parameter gets mapped toward the minimum of f, but the movement is penalized by a squared L_2-norm, resulting in a tradeoff between minimization and remaining near the current estimate. Note that the term being min-

imized in (3.1) is a majorizing function so therefore a prox step is an MM step. One popular proximal algorithm is the proximal gradient method that is designed for minimizing objective functions of the form

$$\ell(\theta) = f(\theta) + g(\theta), \tag{3.2}$$

where f is convex and differentiable and g is convex. The proximal gradient method update alters the objective with a quadratic approximation of the first component f around the current iterate $\theta^{(t-1)}$:

$$\begin{aligned}\theta^{(t)} &= \arg\min_{\theta} \left\{ f(\theta^{(t-1)}) + \nabla f(\theta^{(t-1)})^T(\theta^{(t-1)} - \theta) + \frac{1}{\gamma_t}\|\theta - \theta^{(t-1)}\|^2 + g(\theta) \right\} \\ &= \text{prox}_{\gamma_t g}[\theta^{(t-1)} - \gamma_t \nabla f(\theta^{(t-1)})].\end{aligned} \tag{3.3}$$

Recall the online learning setup of the previous chapter in which the objective takes the form

$$\arg\min_{\theta} \mathbf{E}_Y[\ell(Y, \theta)], \tag{3.4}$$

where the expectation cannot be evaluated, but the learner has access to a sequence of random samples $\{y_t\}_{t=1}^{\infty}$ from the random variable Y. A learner creates a new estimate $\theta^{(t)}$ from the sample y_t and the previous estimate $\theta^{(t-1)}$ using the information contained in $\ell_t(\theta) = \ell(y_t, \theta)$.

In the last several years there has been several developments in adapting the proximal gradient method to stochastic approximation (Duchi and Singer, 2009; Duchi et al., 2011; Bertsekas, 2011; Atchade et al., 2014; Nitanda, 2014; Rosasco et al., 2014). A proximal algorithm that has received less attention in the stochastic approximation setting is the proximal iteration, or proximal point algorithm (Rockafellar, 1976). For a convex function ℓ, the proximal iteration is

$$\theta^{(t)} = \text{prox}_{\gamma_t \ell}(\theta^{(t-1)}). \tag{3.5}$$

The stochastic version of the update above substitutes noisy objectives ℓ_t for ℓ. As the step size γ_t decreases, the L_2 penalty of the prox operator increases and the movement of the update becomes more restricted.

Ryu and Boyd (2014) and Toulis and Airoldi (2015) both study the stochastic proximal iteration (SPI), referred to by the latter as *implicit stochastic gradient descent*. The name *implicit* comes from the fact the update can be interpreted as a variant of stochastic gradient descent (SGD) where $\theta^{(t)}$ appears on both sides of the update equation:

$$\begin{aligned} \theta^{(t)} &= \text{prox}_{\gamma_t \ell_t}(\theta^{(t-1)}) \\ &= \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t)}). \end{aligned} \tag{3.6}$$

Therefore the SPI update does not necessarily have a closed form but it defines an equation to be solved. The main advantage of SPI over SGD is stability (Ryu and Boyd, 2014; Toulis and Airoldi, 2015). SPI updates are more robust to the step size sequence $\{\gamma_t\}_{t=1}^{\infty}$, as SGD error bounds can grow exponentially for a finite number of updates (Bach and Moulines, 2011).

SPI has two major limitations that prevent its use in many cases. The first is that it is limited to convex problems and only certain classes of nonconvex problems (Kaplan and Tichatschke, 1998) for which the proximal mapping is defined. The second limitation is that the proximal mapping for a given objective may be expensive to evaluate. If a prox step does not have a closed form, an iterative procedure is unfortunately necessary inside each update.

In this chapter, we present an algorithm that maintains the stability properties of SPI, yet does not suffer the same limitations. Our method, MSPI, works for both convex and nonconvex problems and offers efficient closed-form updates in many of the cases for which SPI requires an iterative procedure. For example, if the objective is a function of a linear predictor, $\ell_t(\theta) = f_t(x_t^T \beta)$, Toulis and Airoldi (2015) derive updates where a root finding method is required inside of each iteration. We show that MSPI has efficient closed-form updates for many models of this form, including linear regression, logistic regression, distance weighted discrimination, and quantile regression.

3.2 Majorized Stochastic Proximal Iteration

Let $h_t(\theta) = h(y_t, \theta | \theta^{(t-1)})$ majorize $\ell_t(\theta) = \ell(y_t, \theta)$ at $\theta^{(t-1)}$. The Majorized Stochastic Proximal Iteration (MSPI, pronounced “M-Spy”) is defined as a proximal iteration on a

stochastic majorizing function h_t:

$$\boxed{\begin{aligned}\theta^{(t)} &= \text{prox}_{\gamma_t h_t}(\theta^{(t-1)}) \\ &= \theta^{(t-1)} - \gamma_t \nabla h_t(\theta^{(t)}).\end{aligned}} \tag{3.7}$$

MSPI shares the stability properties of SPI but can be applied to a wider class of problems due to the fact that only the majorizer needs a defined proximal mapping; MSPI can readily be applied to non-convex problems as long as the majorizing functions are convex. Also, If the objective function has an expensive proximal mapping, the objective can be swapped out for a majorizing function with a cheap mapping. Rather than deriving an iterative method for calculating a costly prox step, one should instead focus on deriving a majorizing function that makes the prox step trivial.

MSPI via Quadratic Upper Bound

Since h_t can be any kind of majorizing function, the general update does not have a closed form. However, a closed form can be derived in the case of a quadratic upper bound, a popular form of majorization (Hunter and Lange, 2004). Let ℓ_t be twice differentiable and suppose there exists a positive definite matrix A_t such that $A_t - d^2\ell_t(\theta)$ is positive semi-definite for all t. We then get the majorization

$$\begin{aligned}\ell_t(\theta) &\leq \ell_t(\theta^{(t-1)}) + \nabla\ell_t(\theta^{(t-1)})^T(\theta - \theta^{(t-1)}) + \frac{1}{2}(\theta - \theta^{(t-1)})^T a_t(\theta - \theta^{(t-1)}) \\ &= \frac{1}{2}\theta^T A_t\theta + [\nabla\ell_t(\theta^{(t-1)}) - A_t\theta^{(t-1)}]^T\theta + c_t,\end{aligned} \tag{3.8}$$

where c_t contains terms which do not include θ. The MSPI update is to perform a proximal mapping on the right hand side of the above equation. General quadratic functions have a closed-form proximal mapping:

$$\begin{aligned}\ell(\theta) &= \frac{1}{2}\theta^T A\theta + b^T\theta + c, \\ \text{prox}_{\gamma\ell}(\theta) &= (I + \gamma A)^{-1}(\theta - \gamma b).\end{aligned} \tag{3.9}$$

Applying the proximal mapping to the quadratic upper bound, henceforth referred to

as MSPI-Q, we get the update

$$\begin{aligned}\theta^{(t)} &= \text{prox}_{\gamma_t h_t}(\theta^{(t-1)}) \\ &= (I + \gamma_t A_t)^{-1}[\theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)}) + \gamma_t A_t \theta^{(t-1)}] \\ &= (I + \gamma_t A_t)^{-1}[(I + \gamma_t A_t)\theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)})] \\ &= \theta^{(t-1)} - \gamma_t (I + \gamma_t A_t)^{-1} \nabla \ell_t(\theta^{(t-1)}).\end{aligned} \tag{3.10}$$

Note that MSPI-Q looks very similar to SGD, but scales the gradient by an inverted matrix which converges to the identity. A matrix inverse is not an appealing attribute of an online algorithm. However, if A_t is diagonal for all t, MSPI-Q is essentially an element-wise adaptive learning rate algorithm with step size of the form

$$\gamma_{tj} = \frac{\gamma_t}{1 + \gamma_t (A_t)_{jj}}.$$

Algorithms such as AdaGrad (Duchi et al., 2011) also incorporate the inverse of a diagonal matrix, but require the addition of a small constant to ensure that the denominator is nonzero. In contrast, the MSPI-Q "adaptive learning rate" will be well-defined without any such addition.

3.2.1 Regularization

In machine learning problems, regularization methods are used to prevent overfitting the data. Thus, the objectives $\ell_t(\theta)$ have the form

$$\ell_t(\theta) = f_t(\theta) + \lambda \psi(\theta), \tag{3.11}$$

where $\lambda > 0$ is a tuning parameter that controls the amount of regularization and $\psi(\theta)$ is a penalty or regularization term. Some examples of regularization functions are:

$$\begin{aligned}&\psi(\theta) = \|\theta\|_1 \quad \text{(Lasso)}, \\ &\psi(\theta) = \|\theta\|_2^2 \quad \text{(Ridge)}, \\ &\psi(\theta) = (1-\alpha)\|\theta\|_2^2 + \alpha\|\theta\|_1, \alpha \in (0,1) \quad \text{(Elastic Net)}.\end{aligned} \tag{3.12}$$

Composite objectives like (3.11) are easily handled by MSPI, as majorizing func-

tions are often used to split parameters into a separable sum. When parameters are separated as such, deriving the prox operator becomes a univariate problem. As an example, consider a regularized quadratic majorization where $A_t - d^2 f_t(\theta)$ is positive semi-definite for all t:

$$h_t(\theta) = f_t(\theta^{(t-1)}) + g_{tj}^T(\theta - \theta^{(t-1)}) + (\theta - \theta^{(t-1)})^T A_t(\theta - \theta^{(t-1)}) + \lambda\psi(\theta). \tag{3.13}$$

The majorization h_t splits the parameters if A_t is a diagonal matrix. In this case, MSPI-Q can be interpreted as a stochastic proximal gradient method with element-wise learning rates. This result is provided by the following theorem.

Theorem 3.1: MSPI-Q with Diagonal Hessian Approximation

The regularized MSPI-Q update with matrix $A_t = \text{diag}(a_{t1}, \dots a_{td})$, separable regularization function $\psi(\theta) = \sum_{j=1}^{d} \psi_j(\theta_j)$ where each component has a defined proximal mapping, and tuning parameter $\lambda > 0$ is

$$\theta_j^{(t)} = \text{prox}_{\gamma_{tj}\lambda\psi_j}\left[\theta_j^{(t-1)} - \gamma_{tj}\nabla\ell_t(\theta^{(t-1)})_j\right], \tag{3.14}$$

for $\gamma_{tj} = \gamma_t/(1 + \gamma_t a_{tj})$.

Proof. Let $g_{tj} = [\nabla f_t(\theta^{(t-1)})]_j$ and $u = \theta^{(t-1)}$. Then

$$\begin{aligned}
&\text{prox}_{\gamma_t h_t}(\theta^{(t-1)})_j \\
&= \arg\min_{\theta_j}\left\{g_{tj}(\theta_j - u) + \frac{a_{tj}}{2}(\theta_j - u)^2 + \lambda\psi_j(\theta_j) + \frac{1}{2\gamma_t}(\theta_j - u)^2\right\} \\
&= \arg\min_{\theta_j}\left\{g_{tj}\theta_j + \lambda\psi_j(\theta_j) + \left(\frac{1}{2\gamma_t} + \frac{a_{tj}}{2}\right)(\theta_j - u)^2\right\} \\
&= \arg\min_{\theta_j}\left\{g_{tj}\theta_j + \lambda\psi_j(\theta_j) + \frac{1 + \gamma_t a_{tj}}{2\gamma_t}(\theta_j - u)^2\right\} \\
&= \arg\min_{\theta_j}\left\{\lambda\psi_j(\theta_j) + \frac{1 + \gamma_t a_{tj}}{2\gamma_t}\left(\theta_j - u + \frac{\gamma_t}{1 + \gamma_t a_{tj}}g_{tj}\right)^2\right\} \\
&= \arg\min_{\theta_j}\left\{\psi_j(\theta_j) + \frac{1}{2\lambda\gamma_{tj}}\left(\theta_j - u + \gamma_{tj}g_{tj}\right)^2\right\} \\
&= \text{prox}_{\gamma_{tj}\lambda\psi_j}\left(u - \gamma_{tj}g_{tj}\right).
\end{aligned} \tag{3.15}$$

□

3.3 Asymptotic Analysis

In this section we provide proof of convergence to a critical point for a smooth nonconvex objective along with asymptotic normality results for MSPI-Q. Before discussing the asymptotic properties of MSPI, we first list some of the properties of proximal mappings.

Proposition 3.2: Prox Operator Properties

(a) The proximal mapping for a convex function f is *nonexpansive*:

$$\|\text{prox}_f(\theta_1) - \text{prox}_f(\theta_2)\| \leq \|\theta_1 - \theta_2\|. \tag{3.16}$$

(b) If f is μ-strongly convex, the proximal mapping is a *contraction*:

$$\|\text{prox}_f(\theta_1) - \text{prox}_f(\theta_2)\| \leq \frac{1}{1+\mu}\|\theta_1 - \theta_2\|. \tag{3.17}$$

(c) If $\theta_2 = \text{prox}_f(\theta_1)$, then $\theta_2 - \theta_1 \in \partial f(\theta_2)$, where $\partial f(\theta_2)$ is the set of subgradients at θ_2. If f is continous and differentiable, then $\theta_2 = \theta_1 - \nabla f(\theta_2)$.

(d) If f is separable, $f(\theta) = \sum_{j=1}^{d} f_j(\theta_j)$, then

$$[\text{prox}_f(\theta)]_j = \text{prox}_{f_j}(\theta_j), \quad j = 1, \ldots, d. \tag{3.18}$$

Assumption 3.3: MSPI Assumptions

(a) The step size sequence $\{\gamma_t\}_{t=1}^{\infty}$ satisfies $\sum_{t=1}^{\infty} \gamma_t = \infty$, $\sum_{t=1}^{\infty} \gamma_t^2 < \infty$.

(b) $h_t(\theta)$ is M-strongly convex and has an L-Lipschitz continuous gradient. Let $g_t = \nabla h_t(\theta^{(t-1)}), \delta_t = \nabla h_t(\theta^{(t)})$. There exists a constant $B > 0$ such that $\|g_t\|^2 \leq B$.

(c) The objective $\ell(\theta)$ is differentiable, bounded below, and has an R-Lipschitz continuous gradient.

(d) The parameter space is a convex open subset $\Theta \subset \mathbb{R}^d$. $\arg\min_\theta h_t(\theta) \in \Theta$ for all t, data sequence $\{y_t\}_{t=1}^{\infty}$ is independent and identically distributed, and $\ell(y, \theta)$ is well defined for all $(y, \theta) \in \mathcal{Y} \times \Theta$.

Assumption 3.3 (a) is the standard condition on learning rates that step sizes shrink to zero but at not too fast a rate. The conditions in (b) are regularity assumptions about the majorizing function and stochastic gradients that are commonly satisfied in practice. Assumptions (c) and (d) put regularity conditions on the objective function and ensure

the iterations always remain inside the parameter space.

Similar to OMAS and OMAP in the previous chapter, convergence of MSPI relies on nonnegative almost supermartingale theory, as discussed in Robbins and Siegmund (1985). The main result used for MSPI convergence is presented as the following lemma.

Lemma 3.4: Almost Supermartingale Convergence

Suppose there exists sequences of nonnegative random variables A_t, B_t, and C_t, adapted to filtration $\mathcal{F}_t$, such that for all t,

$$\mathbf{E}[M_{t+1}|\mathcal{F}_t] \leq (1 + A_t)M_t - B_t + C_t. \tag{3.19}$$

If $\sum A_t < \infty$ and $\sum C_t < \infty$ almost surely, then M_t converges to a finite limit and $\sum B_t < \infty$ almost surely.

To set notation for the convergence results, define the natural filtration $\mathcal{F}_{t-1}$ as the σ-field generated by $\theta^{(0)}, y_1, y_2, \ldots, y_{t-1}$. Also, let $g_t = \nabla h_t(\theta^{(t-1)}) \in \partial \ell_t(\theta^{(t-1)})$ denote the stochastic subgradients. Note that if $\ell_t(\theta)$ is differentiable, $\nabla \ell_t(\theta^{(t-1)}) = \nabla h_t(\theta^{(t-1)})$ even though $\nabla \ell_t(\theta) \neq \nabla h_t(\theta)$ in general. Let $\delta_t = \nabla h_t(\theta^{(t)})$, so that MSPI updates can be written as

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \delta_t. \tag{3.20}$$

Lemma 3.5: Bound on MSPI Iterates

MSPI iterates satisfy

$$\|\theta^{(t)} - \theta^{(t-1)}\|^2 \leq \gamma_t^2 \|g_t\|^2, \tag{3.21}$$

and therefore $\theta^{(t)}$ converges to some finite value $\theta^{(\infty)}$ almost surely.

Proof. By definition of the proximal mapping,

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_t \nabla h_t(\theta^{(t)}) = \theta^{(t-1)} - \gamma_t \delta_t. \tag{3.22}$$

A slight rearrangement reveals the identity

$$\delta_t = \frac{\theta^{(t-1)} - \theta^{(t)}}{\gamma_t}. \tag{3.23}$$

Since $h_t(\theta)$ is convex, $\nabla h_t(\theta)$ is a monotone mapping which satisfies

$$\begin{aligned}
0 &\leq [g_t - \delta_t]^T (\theta^{(t-1)} - \theta^{(t)}) \\
&\leq \left[g_t - \left(\frac{\theta^{(t-1)} - \theta^{(t)}}{\gamma_t} \right) \right]^T (\theta^{(t-1)} - \theta^{(t)}) \\
&= g_t^T (\theta^{(t-1)} - \theta^{(t)}) - \frac{1}{\gamma_t} \|\theta^{(t-1)} - \theta^{(t)}\|^2 \\
&\leq \|g_t\| \|\theta^{(t-1)} - \theta^{(t)}\| - \frac{1}{\gamma_t} \|\theta^{(t-1)} - \theta^{(t)}\|^2 \quad \text{(Cauchy-Schwarz Inequality)}.
\end{aligned} \tag{3.24}$$

A few steps of basic algebra results in the inequality

$$\gamma_t \|\delta_t\| = \|\theta^{(t-1)} - \theta^{(t)}\| \leq \gamma_t \|g_t\|. \tag{3.25}$$

Since $\sum \gamma_t^2 \|g_t\|^2 \leq \sum \gamma_t^2 B < \infty$, then

$$\sum \|\theta^{(t-1)} - \theta^{(t)}\|^2 < \infty.$$

Therefore $\theta^{(t)} \to \theta^{(\infty)}$ as $t \to \infty$.

□

Lemma 3.6: Asymptotic Equivalence to SGD

$\|\nabla h_t(\theta^{(t)}) - \nabla \ell_t(\theta^{(t-1)})\| = \|\delta_t - g_t\| \to 0$ as $t \to \infty$ almost surely.

Proof. h_t has an L-Lipschitz continuous gradient. By definition,

$$\|g_t - \delta_t\| \leq L\|\theta^{(t-1)} - \theta^{(t)}\| \leq \gamma_t L\|g_t\|. \tag{3.26}$$

□

The above lemmas show that MSPI updates are asymptotically equivalent to SGD and that iterates converge to something. The following theorem states the value converged to is in some sense optimal.

3.3.1 Convergence for Smooth Nonconvex Objective

Theorem 3.7: Convergence for Smooth Nonconvex Objective

Under assumption 3.3, MSPI iterates converge to a critical point of $\ell(\theta)$ almost surely.

Proof. Let $M_t = \ell(\theta^{(t)})$. Without loss of generality, assume $\ell(\theta) \geq 0$ for all θ. Then

$$\begin{aligned} M_t &= \ell(\theta^{(t)}) \\ &\leq \ell(\theta^{(t-1)}) + \nabla\ell(\theta^{(t-1)})^T(\theta^{(t)} - \theta^{(t-1)}) + \frac{R}{2}\|\theta^{(t)} - \theta^{(t-1)}\|^2 \\ &= M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T g_t - \gamma_t \nabla\ell(\theta^{(t-1)})^T(\delta_t - g_t) + \frac{\gamma_t^2 R}{2}\|\delta_t\|^2 \\ &\leq M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T g_t + \gamma_t \|\nabla\ell(\theta^{(t-1)})\|\|\delta_t - g_t\| + \frac{\gamma_t^2 RB}{2} \\ &\leq M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T g_t + \gamma_t^2 L\|\nabla\ell(\theta^{(t-1)})\|\|g_t\| + \frac{\gamma_t^2 RB}{2} \\ &\leq M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T g_t + \gamma_t^2 LB + \frac{\gamma_t^2 RB}{2}, \end{aligned} \tag{3.27}$$

where the last two inequalities use the assumption that $\|g_t\|$ is bounded by $\sqrt{B}$ and lemma 3.6, respectively. Taking expectation with respect to natural filtration $\mathcal{F}_{t-1}$, we get

$$\begin{aligned} \mathbf{E}(M_t|\mathcal{F}_{t-1}) &\leq M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T \mathbf{E}(g_t|\mathcal{F}_{t-1}) + \gamma_t^2 L\sqrt{B} + \frac{\gamma_t^2 RB}{2} \\ &\leq M_{t-1} - \gamma_t \nabla\ell(\theta^{(t-1)})^T \ell(\theta^{(t-1)}) + \gamma_t^2 L\sqrt{B} + \frac{\gamma_t^2 RB}{2}. \end{aligned} \tag{3.28}$$

Then $M_t \geq 0$, the second term is negative , and the last two terms are summable. By lemma 3.4, $\sum_{t=1}^{\infty} \gamma_t \|\nabla\ell(\theta^{(t-1)})\|^2 < \infty$. Since $\sum \gamma_t = \infty$, it must be that

$$\lim_{t\to\infty} \|\nabla\ell(\theta^{(t-1)})\|^2 = \|\nabla\ell(\theta^{(\infty)})\|^2 = 0. \tag{3.29}$$

□

3.4 Nonasymptotic Analysis

3.4.1 Stability

In their discussion of the stability of SPI, Ryu and Boyd (2014) examine the process of minimizing a deterministic function $f_t(x) = x^2/2$, for which MSPI can use the majorizing function

$$h_t(x) = f_t(x) + 1/2\|x - x^{(t-1)}\|^2. \tag{3.30}$$

Updates for SGD, SPI, and MSPI can then be derived as

$$\begin{aligned} \theta_{SGD}^{(t)} &= \theta_{SGD}^{(t-1)} - \gamma_t \theta_{SGD}^{(t-1)}, \\ \theta_{SPI}^{(t)} &= \frac{1}{1+\gamma_t}\theta_{SPI}^{(t-1)}, \\ \theta_{MSPI}^{(t)} &= \frac{1+\gamma_t}{1+2\gamma_t}\theta_{MSPI}^{(t-1)}. \end{aligned} \tag{3.31}$$

It is clear that SGD (gradient descent in the deterministic case) iterates are not guaranteed to decrease the objective; $\theta_{SGD}^{(t)} = 1$ and $\gamma_t > 2$ will move further away from the minimizer 0. In contrast, SPI and MSPI updates are guaranteed to move the parameter towards the optimal value for any $\gamma_t > 0$. SPI updates converge faster than MSPI in this deterministic case, intuitively due to the "extra curvature" in the majorizing function that MSPI is based on.

In the stochastic setting, SPI and MSPI satisfy the stochastic descent property. In other words, updates will always decrease the value of the objective function evaluated on the current observation. The result is trivial by definition of proximal mappings and the proof is omitted from the following proposition. In contrast, stochastic gradient algorithms can only claim that for a decreasing step size γ_t and an objective with an L-Lipschitz continuous gradient, there exists some T such that for all $t \geq T$, updates will decrease the objective value (as discussed in chapter 2).

Proposition 3.8: Stochastic Descent Property

SPI/MSPI iterates satisfy the stochastic descent property for any positive step size $\gamma_t \in (0, 1)$:

$$\ell(y_t, \theta^{(t)}) \leq \ell(y_t, \theta^{(t-1)}). \tag{3.32}$$

3.4.2 Finite Sample Bounds

The following two theorems show that, after redefining a constant, the SPI error bounds are identical to those of MSPI.

Theorem 3.9: SPI Finite Sample Bound with Fixed Step Size

(Ryu and Boyd, 2014). Let ℓ_t be M-strongly convex with unique minimizer θ^* and define

$$\sigma_t = \|\mathsf{prox}_{\gamma_t \ell_t}(\theta^*) - \theta^*\|. \tag{3.33}$$

Then SPI iterates satisfy

$$\|\theta^{(t)} - \theta^*\| \leq \left(\frac{1}{1+\gamma M}\right)^t \|\theta^{(0)} - \theta^*\| + \sup_{i \leq t}(\sigma_i) \left(\frac{1+\gamma M}{\gamma M}\right). \tag{3.34}$$

Theorem 3.10: MSPI Finite Sample Bound with Fixed Step Size

Let ℓ_t be M-strongly convex with unique minimizer θ^* and define

$$\sigma_t = \|\mathsf{prox}_{\gamma_t h_t}(\theta^*) - \theta^*\|. \tag{3.35}$$

Under assumption 3.3 (b), SPI updates satisfy

$$\|\theta^{(t)} - \theta^*\|^2 \leq \left(\frac{1}{1+\gamma M}\right)^t \|\theta^{(0)} - \theta^*\| + \sup_{i \leq t}(\sigma_i)\left(\frac{1+\gamma M}{\gamma M}\right). \tag{3.36}$$

Proof.

$$\begin{aligned}
S_t &= \|\theta^{(t)} - \theta^*\| \\
&= \|\mathsf{prox}_{\gamma h_t}(\theta^{(t-1)}) - \mathsf{prox}_{\gamma h_t}(\theta^*) + \mathsf{prox}_{\gamma h_t}(\theta^*) - \theta^*\| \\
&\leq \left(\frac{1}{1+\gamma M}\right) S_{t-1} + \|\mathsf{prox}_{\gamma h_t}(\theta^*) - \theta^*\| \\
&= \left(\frac{1}{1+\gamma M}\right)^t S_0 + \sum_{i=1}^{t}\left(\frac{1}{1+\gamma M}\right)^{i-1}\sigma_i \\
&\leq \left(\frac{1}{1+\gamma M}\right)^t S_0 + \sup_{i \leq t}(\sigma_i)\sum_{i=0}^{\infty}\left(\frac{1}{1+\gamma M}\right)^i \\
&\leq \left(\frac{1}{1+\gamma M}\right)^t S_0 + \sup_{i \leq t}(\sigma_i)\left(\frac{1+\gamma M}{\gamma M}\right),
\end{aligned} \tag{3.37}$$

where the last inequality comes from properties of geometric series:

$$\sum_{i=0}^{\infty}\left(\frac{1}{1+\gamma M}\right)^i = \frac{1+\gamma M}{\gamma M}. \tag{3.38}$$

□

Finite Sample Bounds for MSPI-Q

Since MSPI-Q can be expressed in closed form, several useful properties can be derived.

Proposition 3.11: Properties of MSPI-Q

1. Mapping for arbitrary θ

$$\text{prox}_{\gamma_t h_t}(\theta) = (I + \gamma_t A_t)^{-1}[\theta + \gamma_t A_t \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)})]. \tag{3.39}$$

2. Difference between mappings

$$\text{prox}_{\gamma_t h_t}(\theta_1) - \text{prox}_{\gamma_t h_t}(\theta_2) = (I + \gamma_t A_t)^{-1}(\theta_1 - \theta_2). \tag{3.40}$$

3. L2-Norm of difference between mappings

$$\|\text{prox}_{\gamma_t h_t}(\theta_1) - \text{prox}_{\gamma_t h_t}(\theta_2)\|^2 = (\theta_1 - \theta_2)^T (I + \gamma_t A_t)^{-2}(\theta_1 - \theta_2). \tag{3.41}$$

4. Difference between parameter and its mapping.

$$\begin{aligned}
\text{prox}_{\gamma_t h_t}(\theta) - \theta &= (I + \gamma_t A_t)^{-1}[\theta + \gamma_t A_t \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)})] - \theta \\
&= (I + \gamma_t A_t)^{-1}[\theta - (I + \gamma_t A_t)\theta + \gamma_t A_t \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)})] \\
&= (I + \gamma_t A_t)^{-1}[-\gamma_t A_t \theta + \gamma_t A_t \theta^{(t-1)} - \gamma_t \nabla \ell_t(\theta^{(t-1)})] \\
&= \gamma_t (I + \gamma_t A_t)^{-1}[A_t(\theta^{(t-1)} - \theta) - \nabla \ell_t(\theta^{(t-1)})].
\end{aligned} \tag{3.42}$$

Lemma 3.12: Bounded Norm of PSD Matrix-Vector Product

For a symmetric positive semi-definite matrix $A \in \mathbb{R}^{d \times d}$ and vector $x \in \mathbb{R}^d$,

$$\sigma_{\mathsf{min}}^2 \|x\|_2^2 \leq \|Ax\|_2^2 \leq \sigma_{\mathsf{max}}^2 \|x\|_2^2,$$

where $\sigma_{\mathsf{min}}, \sigma_{\mathsf{max}}$ are the minimum and maximum singular values of A, respectively.

Proof. A has singular value decomposition $A = UDV^T$ where U, V are orthogonal matrices and D is a diagonal matrix of singular values. Then

$$\begin{aligned} \|Ax\| &= \|UDV^T x\| \\ &= \|DV^T x\| \\ &= \|VDV(V^T x)\| \\ &= \|Dx\| \\ &= \sum_{i=1}^{d} \sigma_i^2 x_i^2. \end{aligned} \tag{3.43}$$

where each line to the next is from properties of orthogonal matrices. It is then clear that

$$\sigma_{\mathsf{min}} \sum_{i=1}^{d} x_i^2 \leq \sum_{i=1}^{d} \sigma_i^2 x_i^2 \leq \sigma_{\mathsf{max}}^2 \sum_{i=1}^{d} x_i^2.$$

□

Lemma 3.13: Bounded Singular Values

For positive semi-definite $A \in \mathbb{R}^{d \times d}$, all singular values of $(I + A)^{-1}$ are less than or equal to 1.

Proof. The symmetric matrix A has eigenvalue decomposition $A = Q\Lambda Q^T$ where Q is orthogonal and Λ is diagonal where values are eigenvalues of A. Then

$$\begin{aligned} I + A &= I + Q\Lambda Q^T \\ &= Q(Q^T Q + \Lambda)Q^T \\ &= Q(I + \Lambda)Q^T. \end{aligned} \tag{3.44}$$

Thus, the eigenvalues of $I + A$ are $1 + \lambda_1, \ldots, 1 + \lambda_d$, where the λ_js are the eigenvalues of A. The singular values of $I + A$ are the eigenvalues of

$$(I + A)^T (I + A) = Q(I + \Lambda)Q^T Q(I + \Lambda)Q^T = Q(I + \Lambda)^2 Q^T.$$

Thus, the singular values of $I + A$ are $(1 + \lambda_1)^2, \ldots, (1 + \lambda_d)^2$. The singular values of $(I + A)^{-1}$ are equal to the inverse of singular values of $I + A$, which are all greater than or equal to 1. □

Lemma 3.14: Bounded Norm

For positive semi-definite matrix $A \in \mathbb{R}^{d \times d}$ and vector x,

$$\|(I + A)^{-1} x\| \leq \|x\|. \tag{3.45}$$

Proof. Combine the above two lemmas. □

Theorem 3.15: MSPI-Q Finite Sample Bound with Fixed Step Size

Let $\ell_t(\theta)$ be strongly convex with unique minimizer θ^* and assume the singular values of the hessian approximations A_t used by the MSPI-Q update are bounded for all t. That is, $0 < \sigma_{\min} < \sigma_{tj} < \sigma_{\max} < \infty$, where σ_{tj}'s are the singular values of A_t. Furthermore, assume the distance between the optimal value and the following quasi-newton update is bounded in expectation:

$$\mathbf{E}(\|\theta^{(t-1)} - A_t^{-1} g_t - \theta^*\| | \mathcal{F}_{t-1}) \leq C. \tag{3.46}$$

Then MSPI-Q iterates satisfy

$$\mathbf{E}(\|\theta^{(t)} - \theta^*\| | \mathcal{F}_{t-1}) \leq \left(\frac{1}{1+\gamma\sigma_{\min}}\right)^t \|\theta^{(0)} - \theta^*\| + \frac{\sigma_{\max}}{\sigma_{\min}} C. \tag{3.47}$$

Proof. Let $H_t\theta = \text{prox}_{\gamma_t h_t}(\theta)$ and $M_t = \|\theta^{(t)} - \theta^*\|$. Then

$$\begin{aligned}
M_t &= \|\theta^{(t)} - \theta^*\| \\
&= \|H_t\theta^{(t-1)} - H_t\theta^* + H_t\theta^* - \theta^*\| \\
&\leq \|H_t\theta^{(t-1)} - H_t\theta^*\| + \|H_t\theta^* - \theta^*\| \\
&\leq \left(\frac{1}{1+\gamma_t\sigma_{\min}}\right) M_{t-1} + \gamma_t \|(I+\gamma_t A_t)^{-1}[A_t(\theta^{(t-1)} - \theta^*) - g_t]\| \\
&\leq \left(\frac{1}{1+\gamma_t\sigma_{\min}}\right) M_{t-1} + \frac{\gamma_t}{1+\gamma_t\sigma_{\min}} \|A_t[\theta^{(t-1)} - \theta^* - A_t^{-1} g_t]\| \\
&\leq \left(\frac{1}{1+\gamma_t\sigma_{\min}}\right) M_{t-1} + \frac{\gamma_t\sigma_{\max}}{1+\gamma_t\sigma_{\min}} \|\theta^{(t-1)} - A_t^{-1} g_t - \theta^*\|.
\end{aligned} \tag{3.48}$$

Taking expectation and using constant step size $\gamma_t = \gamma$,

$$\begin{aligned}
\mathbf{E}(M_t | \mathcal{F}_{t-1}) &\leq \left(\frac{1}{1+\gamma\sigma_{\min}}\right) M_{t-1} + \frac{\gamma\sigma_{\max}}{1+\gamma\sigma_{\min}} C \\
&\leq \left(\frac{1}{1+\gamma\sigma_{\min}}\right)^t M_0 + \sum_{t=1}^{\infty} \left(\frac{1}{1+\gamma\sigma_{\min}}\right)^{t-1} \frac{\gamma\sigma_{\max}}{1+\gamma\sigma_{\min}} C \\
&\leq \left(\frac{1}{1+\gamma\sigma_{\min}}\right)^t M_0 + \frac{\sigma_{\max}}{\sigma_{\min}} C.
\end{aligned} \tag{3.49}$$

□

3.5 Examples

Note on Quadratic Upper Bound for Linear Models

For linear models, the loss takes the form $\ell_t(x_t^T\beta) = \ell(y_t, x_t^T\beta)$, where y_t is the response variable and x_t is a vector of predictor variables. The hessian matrix takes the form

$$d^2\ell_t(x_t^T\beta)x_t x_t^T. \tag{3.50}$$

Since the second derivative takes this form, MSPI-Q updates are often easy to derive due to the following proposition.

Proposition 3.16

For $x \in \mathbb{R}^d$ and a $d \times d$ identity matrix I, $x^T x I - xx^T$ is positive semi-definite.

Suppose $d^2\ell_t(x_t^T\beta)$ is bounded by a constant $c_t(x_t, y_t)$ for all β. Then a quadratic majorization can use $A_t = c_t x_t^T x_t I$ as the hessian approximation.

3.5.1 Linear Regression

The linear regression loss function for a single observation is

$$\ell_t(\beta) = \frac{1}{2}(y_t - x_t^T\beta)^2, \tag{3.51}$$

where $y_t \in \mathbb{R}$ is the response and $x_t \in \mathbb{R}^d$ is a vector of predictor variables. This loss is a quadratic function, so the SPI update is straightforward to derive as

$$\beta^{(t)} = \left(I + \gamma_t x_t x_t^T\right)^{-1}(\beta^{(t-1)} + \gamma_t y_t x_t). \tag{3.52}$$

One possible MSPI update is a quadratic majorization with $A_t = x_t^T x_t I$:

$$\beta^{(t)} = \beta^{(t-1)} + \frac{\gamma_t}{1 + \gamma_t x_t^T x_t}(y_t - x_t^T\beta^{(t-1)})x_t. \tag{3.53}$$

By using the Sherman-Morrison identity

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + u^T A^{-1} v}, \tag{3.54}$$

the SPI update can be shown to be the same as the MSPI update, although MSPI makes it clearer that a linear system does not need to be solved.

3.5.2 Logistic Regression

The logistic regression negative loglikelihood, gradient, and hessian for a single observation $y_t \in \{-1, 1\}, x_t \in \mathbb{R}^d$ are

$$\begin{aligned} \ell_t(\beta) &= \ln(1 + e^{y_t x_t^T \beta}), \quad y_t \in \{-1, 1\}, \\ \nabla \ell_t(\beta) &= \frac{y_t x_t}{1 + e^{-y_t x_t^T \beta}}, \\ d^2 \ell_t(\beta) &= \hat{p}_t(\beta)[1 - \hat{p}_t(\beta)] x_t x_t^T, \quad \hat{p}_t(\beta) = \frac{1}{1 + e^{-x_t^T \beta}}. \end{aligned} \tag{3.55}$$

The predicted probability that $y_t = 1$, $\hat{p}_t(\beta)$, is bounded between 0 and 1. Therefore $\hat{p}(1-\hat{p}) \leq 1/4$ and $\frac{1}{4} x_t x_t^T - d^2 \ell_t(\beta)$ is positive semi-definite. Similarly to the quadratic upper bound for linear regression, we also have that $x_t x_t^T/4 - x_t^T x_t/4I$ is positive semidefinite and an MSPI-Q algorithm using $A_t = x_t^T x_t/4I$ is

$$\beta^{(t)} = \beta^{(t-1)} - \frac{\gamma_t}{1 + \gamma_t x_t^T x_t/4} \left(\frac{y_t}{1 + e^{-y_t x_t^T \beta^{(t-1)}}} \right) x_t. \tag{3.56}$$

3.5.3 Distance Weighted Discrimination

An extension of support vector machine's (SVM) L_1-Hinge Loss, the generalized distance weighted discrimination loss is parameterized by a positive constant q and converges to SVM as $q \to \infty$. The loss and its derivative are:

$$\begin{aligned} v_q(u) &= \begin{cases} 1 - u & \text{if } u \leq \frac{q}{q+1} \\ \frac{q^q}{u^q (q+1)^{q+1}} & \text{if } u > \frac{q}{q+1}, \end{cases} \\ v_q'(u) &= \begin{cases} -1 & \text{if } u \leq \frac{q}{q+1} \\ \left[\frac{q}{u(q+1)} \right]^{q+1} & \text{if } u > \frac{q}{q+1}. \end{cases} \end{aligned} \tag{3.57}$$

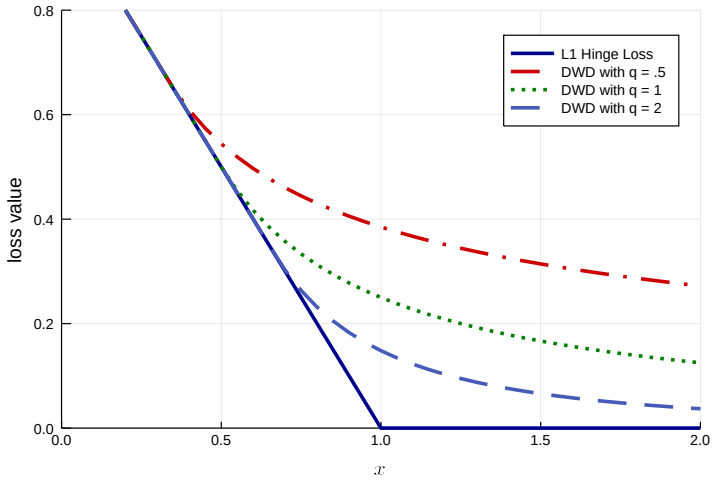


Figure 3.1: DWD Losses compared with Hinge Loss (SVM)

The quadratic majorization discussed in chapter 2 can be used to derive the MSPI-Q update

$$\beta^{(t)} = \beta^{(t-1)} - \frac{\gamma_t}{1 + \gamma_t m_q x_t^T x_t} v_q'(x_t^T \beta) x_t, \tag{3.58}$$

where $m_q = (q+1)^2/q$.

3.5.4 Quantile Regression

The majorizing function for quantile regression described in the previous chapter can be readily applied to MSPI. The quadratic upper bound from Hunter and Lange (2000) can be written for a single observation as

$$h_t(\beta) = \frac{1}{2}\beta^T \frac{x_t x_t^T}{2w_t}\beta - (y_t/(2w_t) + \tau - .5)x_t^T\beta + c_t \tag{3.59}$$

where $w_t = \epsilon + |y_t - x_t^T\beta^{(t-1)}|$ for a small positive constant ϵ. Note that even though this is a quadratic upper bound, it is not derived in the same way as MSPI-Q. Applying the proximal mapping to the above majorization results in MSPI updates

$$\beta^{(t)} = \left(I + \frac{\gamma_t}{2w_t}x_t x_t\right)^{-1}\left\{\beta^{(t-1)} + \gamma_t\left[\frac{y_t}{2w_t} + \tau - .5)\right]x_t\right\}. \tag{3.60}$$

The Sherman-Morrison identity can be applied to remove the need to solve a linear system:

$$\left(I + \frac{\gamma_t}{2w_t} x_t x_t\right)^{-1} = \left(I - \frac{\gamma_t}{2w_t + \gamma_t x_t^T x_t} x_t x_t^T\right). \tag{3.61}$$

3.6 Conclusion

In this chapter, we presented the MSPI algorithm to extend stochastic proximal iteration to a wider class of problems through the use of majorizing functions. MSPI achieves the same robustness as SPI, but works for more objectives and can be implemented more efficiently for linear-in-the-parameter models. Also, existing MM algorithms via a quadratic upper bound can be trivially adapted for MSPI. MSPI achieves the same asymptotic variance of SGD, but has the same stability guarantees of SPI (stochastic descent property). Therefore, in any case where the proximal mapping is readily available, MSPI should be preferred over stochastic gradient methods.

CHAPTER

4

ONLINESTATS.JL: STATISTICS FOR STREAMING AND BIG DATA

4.1 Introduction

The **OnlineStats** package for **Julia** provides methods for calculating statistics and fitting models using online algorithms. An online algorithm accepts input serially so that parameter estimates are updated using only the information from a single observation (or minibatch of observations). Algorithms of this type can naturally handle streaming data and out-of-core (data larger than computer memory) processing. Therefore, where traditional (offline) algorithms are infeasible, online counterparts can be substituted to perform the same statistical analysis. In this chapter we describe a unifying representation of online statistical algorithms that allows **OnlineStats** to rely on the generic programming facilities of **Julia**.

As a trivial example, consider updating a sample mean after adding one new observation x_t. An offline calculation needs to revisit all of the previously observed data,

whereas an online update uses only the new observation and the current estimate:

$$\begin{aligned} \theta^{(t)}_{\text{offline}} &= \frac{1}{t}\sum_{i=1}^{t} x_i, \\ \theta^{(t)}_{\text{online}} &= \left(1 - \frac{1}{t}\right)\theta^{(t-1)}_{\text{online}} + \frac{1}{t}x_t. \end{aligned} \tag{4.1}$$

The complexity of the online update is therefore $O(1)$ compared to $O(t)$ for offline. However, the advantage of being computationally cheap comes at a price. Not every statistic or model can be updated analytically like the mean above and exact solutions may be impossible. In cases like this, **OnlineStats** relies on state-of-the-art stochastic approximation algorithms described in section 4.3.

The goals of **OnlineStats** are broad but few:

1. **OnlineStats** should handle each class of statistical analysis, from summary statistics to generalized linear models.

2. Algorithms should follow an interface that makes it straightforward to add methods.

3. Algorithms should use $O(1)$ memory and be easily adopted to parallel and distributed computing.

At present, there is no direct alternative to **OnlineStats**. While many packages exist for working with big and streaming data, they are typically focused on a small class of problems or type of algorithm. To the author's knowledge, **OnlineStats** is entirely unique in the scope of problems it can solve. For example, packages such as the **Boost Statistical Accumulators Library** [1] and **runstats** [2] provide online updates for various summary statistics, but neither can run a logistic regression. Other packages like **Scikit-Learn** (Pedregosa et al., 2011), **sgd** (Tran and Toulis, 2016), and **gradDescent** (Handian et al., 2017) can run an online logistic regression algorithm, but cannot incrementally calculate a mean or variance. A related package is **biglm** (Lumley, 2013) that implements out-of-core computation of generalized linear models for big data, but through (offline) iterative algorithms.

[1] http://www.boost.org/

[2] http://www.grantjenks.com/docs/runstats/

4.1.1 Why Julia?

The majority of open source statistical software development takes place in **R** (R Core Team, 2017). However, we chose **Julia** (Bezanson et al., 2017) to implement **OnlineStats** for a variety of (mainly technical) reasons. **Julia** achieves impressive performance from a collection of features that work together, such as an expressive type system, just-in-time compilation, multiple dispatch, and metaprogramming tools. Without these tools available, the design of **OnlineStats** would need to be changed dramatically to achieve the same usability and performance. Software in **Julia** is becoming less burdensome to use for users of other languages with the development of interoperability tools like **RCall** [3] for calling **R** from **Julia** and **XRJulia** (Chambers, 2017) for calling **Julia** from **R**. We hope in the near future to have accessible wrapper code available in **R** for using **OnlineStats**.

4.2 Structure of OnlineStats

There are three key types defined in **OnlineStats**: `OnlineStat`, `Weight`, and `Series`. These types correspond to statistics, weighting mechanisms, and collections of statistics applied to a data stream.

4.2.1 `OnlineStat`

Each statistic or model is a subtype of `OnlineStat{N}`, an abstract type parameterized by the "size" of its input. The three common types of input are scalars, vectors, and vector/scalar pairs, which respectively correspond to `OnlineStat{0}`, `OnlineStat{1}`, and `OnlineStat{(1, 0)}`. For example, a single observation for a mean is a scalar, and therefore `Mean` is a subtype of `OnlineStat{0}`. Similarly, a single observation for a covariance matrix is a vector, so `CovMatrix` is a subtype of `OnlineStat{1}`. The interface for working with an `OnlineStat` is defined by three functions:

1. `fit!`
2. `value`

[3] https://github.com/JuliaInterop/RCall.jl

3. `merge!`

This small interface is possible due to **Julia**'s use of multiple dispatch as well as a unifying representation of online updates in which a statistic or model is updated as a function of two things:

1. A new observation.
2. A weight associated with the observation.

Consider the mean example presented in the introduction:

$$\theta^{(t)} = \left(1 - \frac{1}{t}\right)\theta^{(t-1)} + \frac{1}{t}x_t. \tag{4.2}$$

The new observation is x_t and the associated weight is t^{-1}. In **OnlineStats**, the update is generalized for an arbitrary weight $w_t \in [0, 1]$:

$$\theta^{(t)} = (1 - w_t)\theta^{(t-1)} + w_t x_t. \tag{4.3}$$

In terms of the generalized update, we can get an analytical mean by using $w_t = t^{-1}$ or we could calculate an exponentially-weighted mean with $w_t = w$, where $w \in (0, 1)$. Any given `OnlineStat` implements an arbitrary-weight update with the `fit!` function (note that in **Julia**, a function that ends in `!` is a convention that the function *mutates* at least one of the arguments). The `fit!` function always accepts three arguments:

```
fit!(o::TypeOfOnlineStat, x::TypeOfObservation, w::Float64)
```

By **OnlineStats** convention, `fit!` does not necessarily update the parameter directly, but only the "sufficient statistics" for estimating the parameter. This is to avoid unnecessary computations, e.g. updating an `OnlineStat` with observations in a loop and intermediate results are not wanted. To return the estimate, an `OnlineStat` uses the `value` function:

```
value(o::TypeOfOnlineStat)
```

By default, `value` will return the first field of the type (efficiently implemented with **Julia**'s meta-programming features).

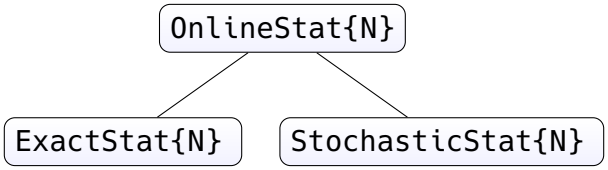
The last piece of the `OnlineStat` interface is `merge!`, which is the basis for parallel computing in **OnlineStats** (discussed in 4.2.3). The `merge!` function is similar to `fit!`, but accepts another `OnlineStat` of the same type instead of a new observation.

```
merge!(o1::TypeOfOnlineStat, o2::TypeOfOnlineStat, w::Float64)
```

ExactStat vs. StochasticStat

There is an important distinction between the two abstract subtypes of `OnlineStat`, shown below in figure 4.1. As the names suggest, an `ExactStat` can be updated with data and merged exactly while a `StochasticStat` uses approximations for both `fit!` and `merge!`. `ExactStat` and `StochasticStat` types also use different default weights, `EqualWeight` and `LearningRate`, which are discussed in section 4.2.2.

Figure 4.1: Abstract OnlineStat Subtypes



Interface Example

The design of **OnlineStats** is heavily based on **Julia**'s use of multiple dispatch. A new `ExactStat` or `StochasticStat` needs only to implement `fit!`, `value` (for which a default is provided), and `merge!` to be completely functional with all features of **OnlineStats**. To demonstrate the ease of creating a new statistic under this simple interface, the entire `Mean` type is defined in **Onlinestats** as follows:

```
mutable struct Mean <: ExactStat{0}
    m::Float64
    Mean(m = 0.0) = new(m)
end

fit!(o::Mean, y::Real, w::Float64) = (o.m += w * (y - o.m))

function Base.merge!(o::Mean, o2::Mean, w::Float64)
  fit!(o, value(o2), w)
end
```

4.2.2 `Weight`

The `Weight` subtypes are used to control the influence of the next observation. Since each `OnlineStat` has updates defined in terms of `fit!(o::OnlineStat, x, w::Float64)`, different weighting mechanisms can generate the weights `w` at each update (section 4.2.3 provides the details of this process). Recall again the mean example where $w_t = t^{-1}$ returns an equally-weighted mean and $w_t = w, w \in (0, 1)$, returns an exponentially-weighted mean. Respectively, **OnlineStats** has `EqualWeight` and `ExponentialWeight` types to provide these weighting schemes. The flexibility of different `Weight` types is designed to handle two situations:

1. **Hyper-parameter tuning for stochastic approximation algorithms** (see section 4.4). The behavior of a given stochastic approximation algorithm may change wildly under different step size sequences (learning rates).

2. **Non-stationarity (parameter drift)**. If the true parameter is changing over time, one can allow a statistic to track to the changes by giving higher weight to new observations.

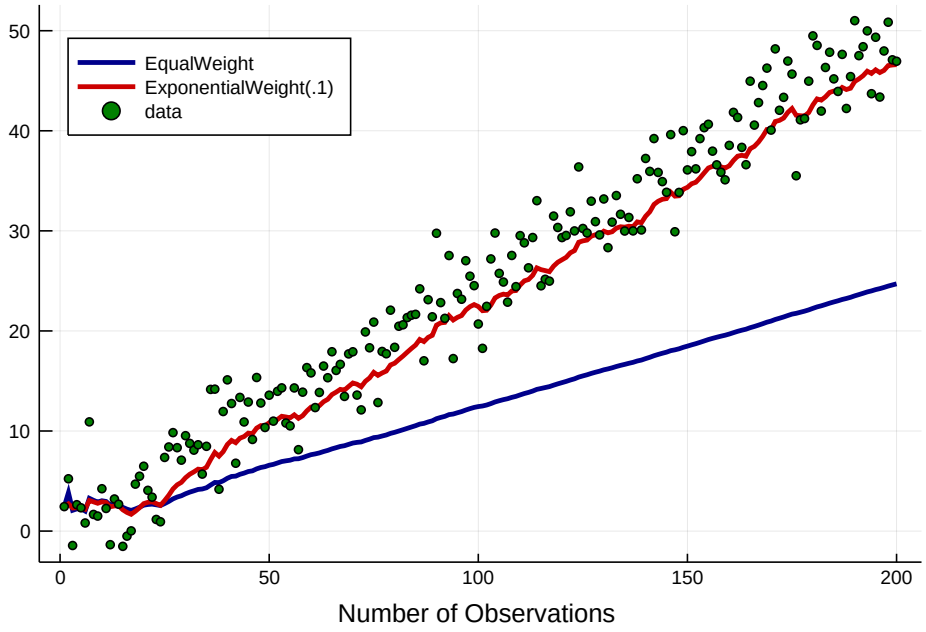


Figure 4.2: Equally-Weighted mean vs. Exponentially-Weighted Mean

Figure 4.2 shows the effect non-stationarity can have on an estimate of the mean. The equally-weighted mean is not able to track the changing data, whereas the exponentially-weighted mean puts higher influence on new observations and is able to track the changes. The phenomenon of the true parameter changing over time can also occur in more complicated statistics and models. **OnlineStats** provides a variety of `Weights` to handle different scenarios. User-defined weighting schemes can also be used with the same high performance as those built into **OnlineStats** (see section 4.5). The full list of built-in weight types is provided below.

- `EqualWeight()`

$$w_t = \frac{1}{t}. \tag{4.4}$$

  Each observation has equal influence. If an `OnlineStat` is capable of an exact analytical update, using `EqualWeight` will provide the same result as the corresponding offline algorithm.

- `ExponentialWeight(c = 0.1)`

$$w_t = c, \quad c \in (0, 1). \tag{4.5}$$

The next observation is given a weight of `c`. The effect is that newer observations have higher influence over the statistic or model than older observations.

- `LearningRate(r = 0.6)`

$$w_t = \frac{1}{t^r}, \quad r > 0. \tag{4.6}$$

Weights decay slower than `EqualWeight`. For $r \in (.5, 1]$, `LearningRate` weights satisfy a common assumption in stochastic approximation that $\sum w_t = \infty, \sum w_t^2 < \infty$. As such, `LearningRate` is designed for use with stochastic approximation algorithms (see section 4.4).

- `LearningRate2(a = 0.5)`

$$w_t = \frac{1}{1 + a(t-1)}, \quad 0 < a < 1. \tag{4.7}$$

As another type aimed at stochastic approximation algorithms, weights decay at a slower rate than `EqualWeight` and satisfy the stochastic approximation assumption described above.

- `HarmonicWeight(a = 10.0)`

$$w_t = \frac{a}{a + t - 1}, \quad a > 0. \tag{4.8}$$

Weights are based on a general harmonic series. Weights decay faster than `EqualWeight` for $a < 1$ and slower for $a > 1$.

- `McclainWeight(c = 0.1)`

$$w_t = \frac{w_{t-1}}{1 + w_{t-1} - c}, \quad c \in (0, 1). \tag{4.9}$$

Weights decay slightly slower than `EqualWeight` and approach `c` in the limit.

Table 4.1: Weight Types Summary

Weight	Step Size	SA Assumption?
`EqualWeight`	$w_t = 1/t$	Yes
`ExponentialWeight`	$w_t = c, c \in (0, 1)$	No
`LearningRate`	$w_t = t^{-r}, r > 0$	If $r \in (0.5, 1]$
`LearningRate2`	$w_t = 1/[1 + a(t-1)], a > 0$	Yes
`HarmonicWeight`	$w_t = a/(a + t - 1), a > 0$	Yes
`McclainWeight`	$w_t = w_{t-1}/(1 + w_{t-1} - c), c \in (0, 1)$	No

Table 4.1 above provides a summary of the `Weight` types in **OnlineStats**. The "SA Assumption?" column is whether the step sizes satisfy $\sum_{t=1}^{\infty} w_t = \infty$, $\sum_{t=1}^{\infty} w_t^2 < \infty$. Figure 4.3 below plots the weight for the first 50 observations. **OnlineStats** implements several *plot recipes* for plotting objects (including `Weight` types) via the **Plots** package [4]. One can quickly compare weight mechanisms by plotting them together:

```
using Plots
plot(EqualWeight())
plot!(ExponentialWeight(.1))
plot!(Bounded(EqualWeight(), .2))
...
```

[4] https://github.com/JuliaPlots/Plots.jl

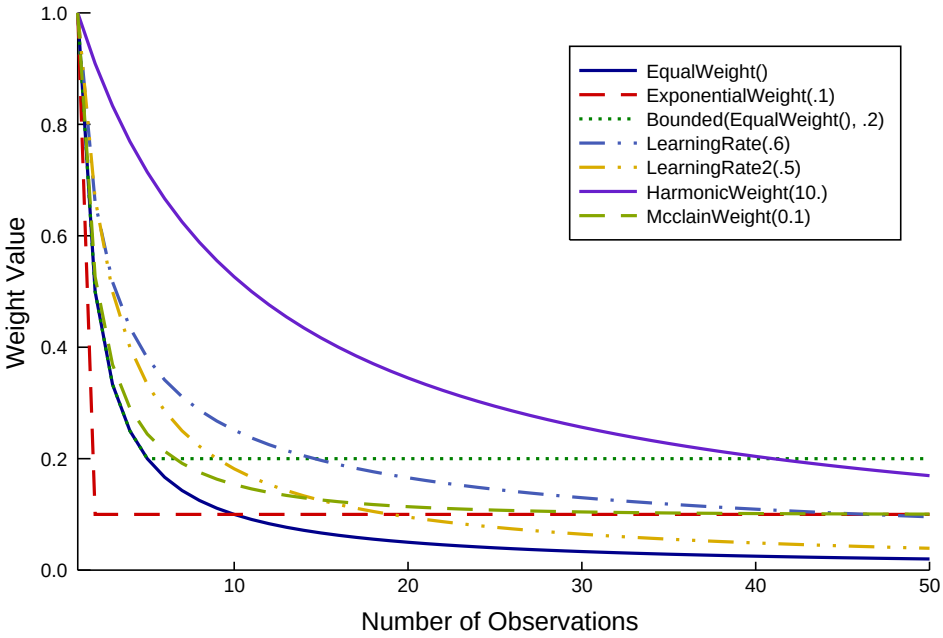


Figure 4.3: Weight Types Visualization

4.2.3 `Series`

The `Series{N}` type is the workhorse of **OnlineStats**, as it represents all of the statistics one wants to apply to a data stream. It keeps track of the number of observations, `Weight`, and any number of `OnlineStats`. The `Series` type is parameterized in the same way as an `OnlineStat{N}`, where `N` defines the "size" of the observation. There are two ways to construct a `Series` (note that in **Julia**, the syntax for a variable number of arguments is `arg::ArgType...`):

1. Start "empty".

```
Series(o::OnlineStat...)
Series(w::Weight, o::OnlineStat...)
```

2. Start with a batch of data

```
Series(data, o::OnlineStats...)
Series(data, w::Weight, o::OnlineStats...)
Series(w::Weight, data, o::OnlineStats...)
```

The type of `data` that can be used in the `Series` constructor depends on the type of `OnlineStat`. **OnlineStats** uses a number of aliases to determine what is allowed:

- `ScalarOb = Union{Number, Symbol, AbstractString}`
- `VectorOb = Union{AbstractVector, Tuple, NamedTuple}`
- `XYOb = Tuple{VectorOb, ScalarOb}`

The `N` parameter of `OnlineStat{N}` then determines which types of data are acceptable inputs:

- For `OnlineStat{0}`, `data` can be a:
  - `ScalarOb` (single observation)
  - `VectorOb` (multiple observations)
- `OnlineStat{1}`, `data` can be a:
  - `VectorOb` (single observation)
  - `AbstractMatrix` (multiple observations)
- `OnlineStat{(1,0)}`, `data` can be a:
  - `XYOb` (single observation)
  - `Tuple{AbstractMatrix, VectorOb}` (multiple observations)

New observations can be given to a `Series` via the `fit!` function. When a new observation is given to a `Series`, the contained `Weight` generates a weight value `w` which then gets passed to the `fit!(o::OnlineStat, y, w)` methods of the contained `OnlineStat` objects. While there is only one `fit!` method per `OnlineStat`, there are several `fit!` methods for `Series`, including updating with a single observation, multiple observations, and optionally overriding the weight:

- **Single observation.**

```
fit!(s::Series, data_i)
```

- **Single observation and override weight.**

  ```
  fit!(s::Series, data_i, w::Float64)
  ```

- **Multiple observations.**

  ```
  fit!(s::Series, data)
  ```

- **Multiple observations and override weight (same weight for all).**

  ```
  fit!(s::Series, data, w::Float64)
  ```

- **Multiple observations and override weight (different weights).**

  ```
  fit!(s::Series, data, w::AbstractVector{Float64})
  ```

Example of Changing State in a Series

The example below shows how the state of objects inside a `Series` change as they get updated with observations. The example begins by constructing a Series (notice the number of observations, `nobs`, starts at zero) with a `Mean`, `Variance`, and `Sum`. When observations of 1 and 2 are passed to the `Series` via `fit!`, the values of each statistic are updated.

```
julia> s = Series(EqualWeight(), Mean(), Variance(), Sum())

    ▩ Series{0}  |  EqualWeight  |  nobs = 0
    ├── Mean(0.0)
    ├── Variance(0.0)
    └── Sum{Float64}(0.0)

julia> fit!(s, 1)

    ▩ Series{0}  |  EqualWeight  |  nobs = 1
    ├── Mean(1.0)
    ├── Variance(0.0)
    └── Sum{Float64}(1.0)
```

```
julia> fit!(s, 2)

    ▩ Series{0}  |  EqualWeight  |  nobs = 2
    ├── Mean(1.5)
    ├── Variance(0.5)
    └── Sum{Float64}(3.0)
```

Merging Series and Parallel Computing

Recall from section 4.2.1 that `OnlineStat` objects can be joined together with the `merge!` function. Similarly, two `Series` can be joined if they track the same types of `OnlineStat`. For an `ExactStat` like `Mean`, `fit!` and `merge!` operations can occur in any order and still return the correct value of the sample mean. This facilitates what is commonly called an *embarrassingly parallel* computation, as the problem of calculating a mean can trivially be split into subproblems that can be worked on simultaneously. The example below and following visualization demonstrates calculating a mean, variance, and histogram on 30,000 observations, where three `Series` each get updated with a third of the data. The `Series` objects can then be merged together, creating an estimate equivalent to a single `Series` that is updated with all 30,000 observations.

Note that an exact merge is only possible if the `OnlineStat` is a subtype of `ExactStat`. For `StochasticStat` subtypes, merging may not be a well-defined operation. In these cases, **OnlineStats** will either print a warning or use a heuristic technique that works in most cases. In general, `merge!` is a more expensive operation than `fit!`. With `StochasticStat` types, `merge!` is also in general a less robust method of updating and thus `fit!` should be preferred.

```
y1 = randn(10_000)
y2 = randn(10_000)
y3 = randn(10_000)

s1 = Series(Mean(), Variance(), IHistogram(50))
s2 = Series(Mean(), Variance(), IHistogram(50))
s3 = Series(Mean(), Variance(), IHistogram(50))

# each series gets one third of the data
```

```
fit!(s1, y1)
fit!(s2, y2)
fit!(s3, y3)

merge!(s1, s2)  # merge the information from s2 into s1
merge!(s1, s3)  # merge the information from s3 into s1
```

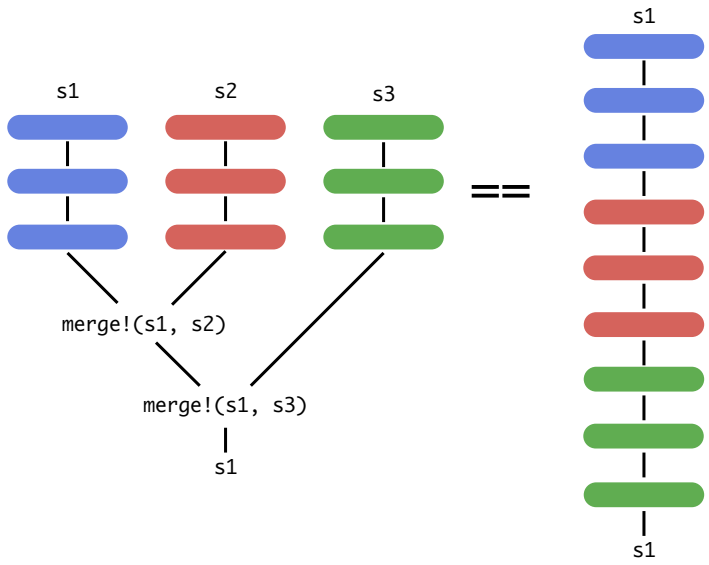


Figure 4.4: Visualization of embarrassingly parallel computations in OnlineStats using the `merge!` function.

The above feature of fitting statistics and models in parallel is what allows **OnlineStats** to work with **JuliaDB** [5] for calculating statistics in a scalable matter on huge datasets. **JuliaDB** implements data table and querying operations for everything from trivial examples to large multiple-file persistent datasets. The integration with **OnlineStats** gives users the ability to calculate statistics and models out-of-core and in parallel.

[5] http://juliadb.org

4.3 Stochastic Approximation Algorithms

For statistics and models that cannot be updated exactly (`StochasticStat{N}`), **OnlineStats** relies on a variety of stochastic approximation algorithms. Each subtype of `StochasticStat` implements at least one of the algorithms in this section. Each algorithm appears as its own type (`Updater`) in **OnlineStats** so that **Julia**'s use of multiple dispatch can create specialized code for each type of update.

The goal in stochastic approximation is to minimize the expected value of a loss function, with respect to an unknown random variable:

$$\arg\min_{\theta} \mathbf{E}_Y[\ell(Y, \theta)]. \tag{4.10}$$

The expectation cannot be evaluated directly since the distribution of Y is unknown, but a learner has access to random samples y_t from Y. For a given random sample y_t, we use the information contained in

$$\ell_t(\theta) = \ell(y_t, \theta) \tag{4.11}$$

to update our parameter $\theta^{(t-1)}$ to $\theta^{(t)}$. Many of the algorithms in this section rely solely on *stochastic gradients*, which we denote as $g_t = \nabla \ell_t(\theta^{(t-1)})$. Let ϵ be a small positive constant (occasionally used to avoid dividing by zero, etc.) and let $w_t > 0$ be an arbitrary weight.

4.3.1 Stochastic Gradient Algorithms

- `SGD`

  Stochastic gradient descent (SGD) is the gold standard of stochastic approximation algorithms, as it has a long history (Robbins and Monro, 1951; Sakrison, 1965) and is easy to implement. The SGD update is

  $$\theta^{(t)} = \theta^{(t-1)} - w_t g_t. \tag{4.12}$$

- `ADAGRAD`

Duchi and Singer (2009) present ADAGRAD as a method for adapting separate learning rates for each parameter element. The algorithm uses the sum of outer product of stochastic gradients to create a matrix that is used to scale the gradients, resulting in element-wise learning rates. ADAGRAD is represented differently in **OnlineStats** compared to the original paper: the outer product matrix is considered to be a *mean* rather than a *sum*, which reveals an implicit learning rate of $w_t = 1/\sqrt{t}$. By identifying the "built-in" learning rate, **OnlineStats** is then able to substitute in any weighting scheme (`Weight`) with `ADAGRAD`.

$$\begin{aligned} G_t &= (1 - 1/t)G_{t-1} + (1/t)\text{diag}(g_t g_t^T + \epsilon I) \\ \theta^{(t)} &= \theta^{(t-1)} - w_t G_t^{-1/2} g_t. \end{aligned} \tag{4.13}$$

- `RMSPROP`

  One of the critiques of ADAGRAD is that the step size may shrink to zero too quickly Ruder (2017). RMSProp (Root mean square propagation) is a popular, yet unpublished idea that first appeared in the Coursera [6] class *Neural Networks for Machine Learning* (Tieleman and Hinton, 2012). The main idea is to use the same *mean* representation of the outer product matrix used by **OnlineStats**, but make it exponentially-weighted. For hyper-parameter $\rho \in (0, 1)$, suggested in Tieleman and Hinton (2012) to be $\rho = .9$, the RMSProp update is

$$\begin{aligned} G_t &= \rho G_{t-1} + (1 - \rho)\text{diag}(g_t g_t^T + \epsilon I), \\ \theta^{(t)} &= \theta^{(t-1)} - w_t G_t^{-1/2} g_t. \end{aligned} \tag{4.14}$$

- `ADADELTA`

  The idea of ADADELTA, presented in Zeiler (2012), is to calculate the conditioning matrix as an exponentially weighted mean (same as RMSProp) and then scale the gradient into the same units as the parameter. For hyper-parameter

[6] https://www.coursera.org

$\rho \in (0, 1)$, the ADADELTA update is

$$\begin{aligned} G_t &= \rho G_{t-1} + (1-\rho)\mathsf{diag}(g_t g_t^T + \epsilon I) \\ \delta_t &= \Delta_{t-1}^{1/2} G_t^{-1/2} g_t \\ \Delta_t &= \rho \Delta_{t-1} + (1-\rho)\mathsf{diag}(\delta_t \delta_t^T + \epsilon I) \\ \theta^{(t)} &= \theta^{(t-1)} - \delta_t, \end{aligned} \tag{4.15}$$

Note that ADADELTA truly does not use a learning rate sequence $\{w_t\}$ and instead step sizes are implicitly defined by the update.

- `ADAM` and `ADAMAX`

  ADAM (adaptive moment estimation) is another method for element-wise learning rates that aims at slowing ADAGRAD's aggressively decreasing learning rate (Kingma and Ba, 2014). The idea is to first calculate an exponentially weighted mean for the outer product matrix (same as for RMSProp and ADADELTA), second to use a gradient with momentum (exponentially weighted gradient estimate), and third make an adjustment for the means since they are biased towards zero.

  The ADAM update is

$$\begin{aligned} m_{tj} &= \beta_1 m_{t-1,j} + (1-\beta_1) g_{tj} \\ v_{tj} &= \beta_2 v_{t-1,j} + (1-\beta_2) g_{tj}^2 \\ \hat{m}_{tj} &= m_{tj}/(1-\beta_1^t) \\ \hat{v}_{tj} &= v_{tj}/(1-\beta_2^t) \\ \theta_j^{(t)} &= \theta_j^{(t-1)} - w_t \frac{\hat{m}_{tj}}{\sqrt{\hat{v}_{tj}} + \epsilon}. \end{aligned} \tag{4.16}$$

  The ADAMAX variant of ADAM alters the denominator to get updates:

$$\begin{aligned} m_{tj} &= \beta_1 m_{t-1,j} + (1-\beta_1) g_{tj} \\ u_{tj} &= \mathsf{max}(\beta_2 u_{t-1,j}, |g_{tj}|) \\ \theta_j^{(t)} &= \theta_j^{(t-1)} - \frac{w_t}{(1-\beta_1^t) u_{tj}} m_{tj}. \end{aligned} \tag{4.17}$$

4.3.2 Stochastic MM Algorithms

Stochastic MM (Majorization-minimization) algorithms rely on *majorizing functions* to inform updates rather than gradients like the algorithms in the previous section. First we discuss the offline MM algorithm, which is a generalization of the expectation-maximization (EM) algorithm (Dempster et al., 1977). A function h is said to *majorize* ℓ at the point θ' if h satisfies:

$$\begin{aligned} h(\theta'|\theta') &= \ell(\theta'), \\ h(\theta|\theta') &\geq \ell(\theta) \quad \text{for all } \theta. \end{aligned} \tag{4.18}$$

In other words, $h(\theta)$ always lies above $\ell(\theta)$, except at θ' where they are equal. The traditional (offline) MM algorithm alternates between two steps:

1. **Majorization**. Generate a function $h(\theta|\theta^{(t-1)})$ that majorizes the objective function at the current iterate $\theta^{(t-1)}$.

2. **Minimization**. Minimize the majorizing function: $\theta^{(t)} = \arg\min_\theta h(\theta|\theta^{(t-1)})$.

The figure below demonstrates one MM iteration applied to a univariate function. The blue line is the objective function and $\theta^{(t-1)}$ is the initial value of the estimate. First, the majorization step creates the red line. Then the minimization step finds the value of θ that minimizes the red function, moving our estimate to $\theta^{(t)}$. This visualization demonstrates the *descent property* of MM algorithms; iterations are guaranteed to decrease the value of the objective function.

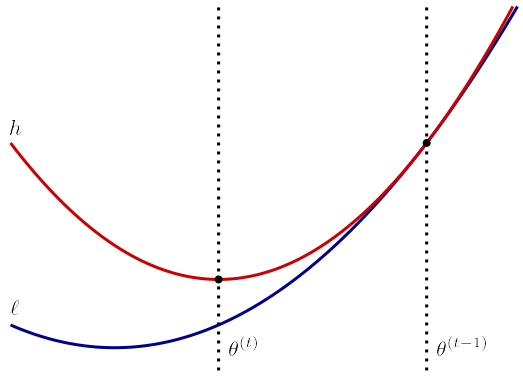


Figure 4.5: Visualization of One MM Iteration

Recall that stochastic approximation algorithms only have access to one observation at a time. Just as the algorithms of the previous section were based on stochastic gradients, the following algorithms are based on *stochastic majorizations*, functions that majorize the objective evaluated at the current observation y_t:

$$h_t(\theta) = h(y_t, \theta|\theta^{(t-1)}), \tag{4.19}$$

where $h_t(\theta)$ majorizes $\ell_t(\theta)$ at $\theta^{(t-1)}$.

MM is more of a concept than an actual algorithm. **OnlineStats** allows for three different "families" of stochastic MM algorithm:

1. **Online MM via Averaged Surrogate**: `OMAS`

   The idea of `OMAS` is to approximate the entire majorizing function, which is referred to here as a "surrogate" objective function. OMAS is introduced under the name *Stochastic Majorization-Minimization* in Mairal (2013b) as an extension to the online EM algorithm in Bach and Moulines (2011). Iterations then minimize the updated approximation:

$$\begin{aligned} Q_t(\theta) &= (1 - w_t)Q_{t-1}(\theta) + w_t h_t(\theta) \\ \theta^{(t)} &= \arg\min_{\theta} Q_t(\theta) \end{aligned} \tag{4.20}$$

2. **Online MM via Averaged Parameter**: `OMAP`

   `OMAP` is different from `OMAS` in that the averaging occurs on minimizers to the stochastic majorizations:

$$\theta^{(t)} = (1 - \gamma_t)\theta^{(t-1)} + \gamma_t \arg\min_{\theta} h_t(\theta) \tag{4.21}$$

3. **Majorized Stochastic Proximal Iteration**: `MSPI`

   There exists a variant of SGD called *implicit stochastic gradient descent* (Toulis et al., 2017), also referred to as a *stochastic proximal iteration* (SPI) (Ryu and Boyd, 2014), in which the update to the parameter appears on both sides of the

update equation (equivalent to a scaled *proximal mapping*):

$$\begin{aligned}\theta^{(t)} &= \theta^{(t-1)} + w_t \nabla \ell_t(\theta^{(t)}) \\ &= \text{prox}_{w_t \ell_t}(\theta^{(t-1)}).\end{aligned} \tag{4.22}$$

SPI achieves higher stability than many stochastic gradient algorithms, but this comes at the cost of needing to solve the above equation at each update, for which there may not be a closed form solution and requires an iterative procedure to solve. The idea of `MSPI` is to replace the objective function with a majorization that has a closed form proximal mapping:

$$\begin{aligned}\theta^{(t)} &= \theta^{(t-1)} - w_t \nabla h_t(\theta^{(t)}) \\ &= \text{prox}_{w_t h_t}(\theta^{(t-1)}).\end{aligned} \tag{4.23}$$

The update can thus avoid an iterative solver but maintains the advantageous stability properties of SPI.

Majorization Via Quadratic Upper Bound

Creating majorizing functions is a bit of an art form. The most common majorization used in **OnlineStats** is given in Hunter and Lange (2004): majorization via a quadratic upper bound. First consider a second-order Taylor series approximation to a function:

$$\ell(\theta) \approx \ell(\theta^{(t)}) + \nabla \ell(\theta^{(t)})^T(\theta - \theta^{(t)}) + \frac{1}{2}(\theta - \theta^{(t)})^T d^2\ell(\theta^{(t)})(\theta - \theta^{(t)}), \tag{4.24}$$

where $d^2\ell(\theta)$ is the second derivative (Hessian) matrix evaluated at θ. If there exists a matrix H such that $H - d^2\ell(\theta)$ is positive semi-definite for all θ, then substituting H for $d^2\ell(\theta^{(t)})$ returns an inequality:

$$\ell(\theta) \leq \ell(\theta^{(t)}) + \nabla \ell(\theta^{(t)})^T(\theta - \theta^{(t)}) + \frac{1}{2}(\theta - \theta^{(t)})^T H(\theta - \theta^{(t)}). \tag{4.25}$$

Note that the right hand side is a majorization of $\ell(\theta)$ at $\theta^{(t)}$. In the stochastic approximation setting, let H_t be a matrix such that $H_t - d^2\ell_t(\theta)$ is positive semi-definite

for all θ. Then the OMAS, OMAP, and MSPI updates can be derived as follows (note that g_t denotes the stochastic gradient from the previous section):

1. OMAS

$$\begin{aligned} A_t &= (1 - w_t)A_{t-1} + w_t H_t, \\ b_t &= (1 - w_t)b_{t-1} + w_t[H_t\theta^{(t-1)} - g_t], \\ \theta^{(t)} &= A_t^{-1} b_t. \end{aligned} \tag{4.26}$$

2. OMAP

$$\theta^{(t)} = \theta^{(t-1)} - w_t H_t^{-1} g_t. \tag{4.27}$$

3. MSPI

$$\theta^{(t)} = \theta^{(t-1)} - w_t(I + w_t H_t)^{-1} g_t. \tag{4.28}$$

Not that for all three of the above updates, if H_t is a diagonal matrix, updates can be executed element-wise.

4.4 Algorithm Catalog

This section contains a comprehensive alphabetical list of the current statistic and model algorithms implemented in **OnlineStats**. The updates to each statistic or model is given in terms of an arbitrary weight

$$w_t \in [0, 1].$$

4.4.1 Covariance Matrix (`CovMatrix`)

The covariance matrix of random variable $Y \in \mathbb{R}^p$ is defined as the expectation

$$\Sigma = \mathbf{E}\left\{[Y - \mathbf{E}(Y)][Y - \mathbf{E}(Y)]^T\right\}. \tag{4.29}$$

The finite sample estimator for a data matrix $X = [y_1\ y_2\ \ldots\ y_n]^T \in \mathbb{R}^{n \times p}$ is

$$\hat{\Sigma} = (n-1)^{-1}(X - \overline{X})^T(X - \overline{X}), \tag{4.30}$$

where $\overline{X}$ is a matrix where each element of column j is equal to the mean of column j of X. That is, $\overline{X} = n^{-1} 1_n 1_n^T X$ where 1_n is a length n vector of ones. With a bit of algebra, the sample covariance matrix can be shown to be

$$\begin{aligned}
\hat{\Sigma} &= \frac{1}{n-1}(X - \overline{X})^T (X - \overline{X}) \\
&= \frac{1}{n-1}\left(X^T X - X^T \overline{X} - \overline{X}^T X - \overline{X}^T \overline{X}\right) \\
&= \frac{1}{n-1}\left(X^T X - X^T 1_n 1_n^T X / n\right) \\
&= \frac{n}{n-1}(A - bb^T),
\end{aligned} \tag{4.31}$$

where $A = X^T X / n$ and $b = X^T 1_n / n$ (vector of column means). Translating the update to arbitrary weights, we get:

$$\begin{aligned}
A^{(t)} &= (1 - \gamma_t) A^{(t-1)} + \gamma_t y_t y_t^T \\
b^{(t)} &= (1 - \gamma_t) b^{(t-1)} + \gamma_t y_t \\
\theta^{(t)} &= \frac{n_t}{n_t - 1}\left\{A^{(t)} - b^{(t)}[b^{(t)}]^T\right\}.
\end{aligned} \tag{4.32}$$

CovMatrix Example

```
julia> Series(randn(1000, 2), CovMatrix(2))
▦ Series{1} with EqualWeight
  ├── nobs = 1000
  └── CovMatrix([1.04641 0.0271401; 0.0271401 0.937007])
```

4.4.2 Differences (`Diff`)

`Diff` tracks the most recent value and the most recent difference, calculated as

$$\theta^{(t)} = y_t - y_{t-1}. \tag{4.33}$$

`Diff` is meant to be used in conjunction with other `OnlineStats`, such as calculating a mean absolute difference. It is parameterized by the data type to avoid loss of information, e.g. converting an `Int64` into a `Float64`.

Diff Example

```
julia> Series([1, 2], Diff(Int))
▦ Series{0} with EqualWeight
  ├── nobs = 2
  └── Diff{Int64}(1)

julia> Series([1.0, 2.0], Diff(Float64))
▦ Series{0} with EqualWeight
  ├── nobs = 2
  └── Diff{Float64}(1.0)
```

4.4.3 Extrema (`Extrema`)

The updates for maximum and minimum are trivially calculated as

$$\begin{aligned} \theta_{\text{max}}^{(t)} &= \max(\theta_{\text{max}}^{(t-1)}, y_t), \\ \theta_{\text{min}}^{(t)} &= \min(\theta_{\text{min}}^{(t-1)}, y_t). \end{aligned} \tag{4.34}$$

Extrema Example

```
julia> Series(randn(1000), Extrema())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── Extrema((-3.76838, 2.67421))
```

4.4.4 Histograms (`Hist`)

OnlineStats uses the `Hist` type to create histograms from data streams. Under the hood, there are two completely separate fitting algorithms:

1. `Hist(b::Integer)`: incrementally build a histogram of `b` bins.

   The idea is for each new observation y_t, create a new bin center/count pair of $(y_t, 1)$, sort all of the bins by the y value, and then merge the two closest bins (Ben-Haim and Tom-Tov, 2010). For a histogram consisting of bin centers

and counts, $(z_1, c_1), \ldots, (z_b, c_b)$, let the first b observations initialize the estimate: $(y_1, 1), \ldots, (y_b, 1)$. For each following observation y_t:

$$\begin{aligned}
&\text{Create pair: } (y_t, 1) \\
&\text{Sort by first value in pair :} (z_1, c_1), \ldots, (y_t, 1), \ldots, (z_b, c_b) \\
&\text{Find } k \text{ that minimizes } z_{k+1} - z_k \\
&\text{Replace } (z_k, c_k) \text{ with } \left(\frac{z_k c_k + z_{k+1} c_{k+1}}{c_k + c_{k+1}}, c_k + c_{k+1} \right)
\end{aligned} \tag{4.35}$$

2. `Hist(r::Range)`: build a histogram over bin edges defined by `r`.

   This algorithm calculates a histogram for bin edges provided by `r`. For a histogram of bins ranging from a to b with bin width δ, the update rule for an observation $a \leq y \leq b$ for the vector of counts $v = v_1, \ldots v_{(b-a)/\delta}$ is

$$\begin{aligned}
k &= \text{floor}[(y_t - a)/\delta] + 1, \\
v_k &\leftarrow v_k + 1.
\end{aligned} \tag{4.36}$$

Algorithm 2 is faster, but requires the user to specify endpoints before the data is observed. Algorithm 1 has the advantage of automatically finding the best bin locations, but at the disadvantage of performance. Any `Hist` type can be used to calculate approximate summary statistics (`mean`, `var`, `std`, `quantile`) without the need to revisit the data.

Histogram Example

```
julia> o1, o2 = Hist(25), Hist(-5:.1:5);

julia> s = Series(randn(100_000), o1, o2);

julia> mean(o1)
-0.002079880278233522

julia> mean(o2)
0.09807099999999999
```

4.4.5 K-Means Clustering (`KMeans`)

K-means clustering is an unsupervised learning technique that assumes each observation comes from one of K clusters. Since each observation is a vector, a `KMeans` object must be constructed with the observation length and number of clusters. As there is no closed-form solution for an online update, the update rule via `SGD` is

$$\begin{aligned} k^* &= \arg\min_k \|y_t - \theta_k\|_2^2 \\ \theta_{k^*}^{(t)} &= (1 - w_t)\theta_{k^*}^{(t-1)} + w_t y_t. \end{aligned} \tag{4.37}$$

KMeans Example

```
julia> y = randn(10_000, 3);

julia> y[rand(Bool, 10_000), :] .+= [1 2 3];

julia> Series(y, KMeans(3, 2))
▦ Series{1}  |  LearningRate(r = 0.6)  |  nobs = 10000
└── KMeans{SGD}([1.0827 -0.0577; 2.0986 0.0094; 3.086 -0.0371])
```

4.4.6 Linear Regression (`LinReg` and `LinRegBuilder`)

For a matrix $X = (x_1, x_2, \dots x_p)^T \in \mathbb{R}^{n \times p}$ (observations in rows) and vector $y = (y_1, \dots, y_n) \in \mathbb{R}^n$, the linear regression model is

$$y = X\beta + \epsilon, \tag{4.38}$$

where $\beta \in \mathbb{R}^p$ is a parameter vector and $\epsilon \in \mathbb{R}^n$ is an uncorrelated error vector with constant variance and mean 0. The linear regression problem (normalized by n) with a ridge penalty is to solve

$$\arg\min_\beta \frac{1}{2n}(y - X\beta)^T(y - X\beta) + \frac{1}{2}\lambda\beta^T\beta. \tag{4.39}$$

Taking the derivative and setting it equal to zero, we find the solution satisfies

$$(X^T X + \lambda I)\beta = X^T y. \tag{4.40}$$

The matrix $X^T X + \lambda I$ and vector $X^T y$ can both be updated analytically. Translating the online update to use arbitrary weights, the update rule is

$$\begin{aligned} A_t &= (1-\gamma_t)A_{t-1} + \gamma_t x_t x_t^T, \\ b_t &= (1-\gamma_t)b_{t-1} + y_t x_t, \\ \beta^{(t)} &= (A_t + \lambda I)^{-1} b_t. \end{aligned} \tag{4.41}$$

The "paper solution" above is not the most efficient way to calculate the parameter in a computer. **OnlineStats** uses the symmetric sweep operator instead of explicitly calculating the matrix inverse. Here we give a brief introduction to the sweep operator and point more interested readers to Lange (2010). Let A be a symmetric matrix. Sweeping on the k-th diagonal entries maps A to a new symmetric matrix $\hat{A}$ such that

$$\begin{aligned} \hat{A}_{kk} &= \frac{-1}{A_{kk}} \\ \hat{A}_{ik} &= \frac{A_{ik}}{A_{kk}} \\ \hat{A}_{kj} &= \frac{A_{kj}}{A_{kk}} \\ \hat{A}_{ij} &= A_{ij} - \frac{A_{ik}A_{kj}}{A_{kk}}. \end{aligned} \tag{4.42}$$

If A is a symmetric block matrix of the form

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{4.43}$$

Sweeping over all diagonal entries of A_{11} results in

$$\begin{pmatrix} A_{11}^{-1} & A_{11}^{-1}A_{12} \\ A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}. \tag{4.44}$$

Then by letting $A_{11} = X^T X + \lambda I$, $A_{12} = X^T y$, and $A_{22} = y^T y$, sweeping on A_{11} returns

$$\begin{pmatrix} (X^T X + \lambda I)^{-1} & \hat{\beta} \\ \hat{\beta}^T & \|y - \hat{y}\|_2^2 \end{pmatrix}. \tag{4.45}$$

The `LinReg` type is for running a linear regression model where it is known which variables are predictors and which variable is the response. A more exploratory approach to model building is the `LinRegBuilder` type, which allows one to use any of the variables as the response and any subset of variables as predictors. The usefulness of `LinRegBuilder` is that the data does not need to be revisited to fit additional models or perform variable selection.

LinReg Example

```
julia> x = randn(1000, 5);

julia> y = x * collect(1.0:5) + randn(1000);

julia> Series((x,y), LinReg(5))
▦ Series{(1, 0)}  |  EqualWeight  |  nobs = 1000
└── LinReg: β(0.0) = [0.955116 2.01755 3.01617 3.98708 4.96303]
```

LinRegBuilder Example

```
julia> xy = [x y];

julia> o = LinRegBuilder(6);

julia> Series(xy, o)
▦ Series{1}  |  EqualWeight  |  nobs = 1000
└── LinRegBuilder of 6 variables

julia> coef(o, y=[6], x=[5,4,3,2,1], bias=false)
5-element Array{Float64,1}:
 4.96303
```

```
3.98708
3.01617
2.01755
0.955116
```

4.4.7 Generalized Linear Models (`StatLearn`)

Generalized linear models (GLMs) cannot be solved analytically with online algorithms apart from linear regression. **OnlineStats** implements stochastic approximation algorithms to handle loss functions from **LossFunctions.jl** [7] and regularization functions from **PenaltyFunctions.jl** [8] via the `StatLearn` type. The (offline) objective function that StatLearn approximately minimizes is

$$\frac{1}{n}\sum_{i=1}^{n} f_i(\beta) + \sum_{j=1}^{p} \lambda_j g(\beta_j), \tag{4.46}$$

where f_i is a loss function evaluated on a single observation, β is the parameter vector, g is a regularization or penalty function to help avoid over-fitting, and λ_j is an element-wise tuning parameter that controls the amount of regularization. This representation of models incorporates much more than GLMs and is often called *empirical risk minimization*. We begin with an example to show the objective function above is represented by `StatLearn`:

```
julia> StatLearn(10, .5L2DistLoss(), L1Penalty(), .1ones(10), SGD())

StatLearn{SGD,ScaledDistanceLoss{LPDistLoss{2},05},L1Penalty}
    > β       : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    > λfactor : [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
    > Loss    : 0.5 * (L2DistLoss)
    > Penalty : L1Penalty
    > Updater : SGD
```

The first argument to the `StatLearn` constructor is the number of predictors variables, followed by arguments to set the loss, penalty, tuning parameters, and algorithm.

[7] https://github.com/JuliaML/LossFunctions.jl
[8] https://github.com/JuliaML/PenaltyFunctions.jl

The arguments can be given in any order and `StatLearn` will use the argument types to determine how they should be handled. Consider each of the arguments provided to `StatLearn` example above:

- `.5 * L2DistLoss()`. This specifies the loss function as

$$f_i(\beta) = \frac{1}{2}(y_i - x_i^T\beta)^2. \tag{4.47}$$

- `L1Penalty()`. This specifies the penalty function as

$$g(\beta_j) = |\beta_j|. \tag{4.48}$$

- `fill(.1, 10)`. This specifies the tuning parameters as

$$\lambda_j = .1, \quad j = 1, \ldots, 10. \tag{4.49}$$

- `SGD()`. This specifies the updater as proximal stochastic gradient descent.

Every updater in section 4.3 is available to use with `StatLearn`. Penalty function are handled with proximal mappings. TODO: finish thought.

StatLearn Example

```
julia> x = randn(1000, 5);

julia> y = x * [1, 2, 3, 4,5] + randn(1000);

julia> Series((x,y), StatLearn(5))
▩ Series{(1, 0)}  |  LearningRate(r = 0.6)  |  nobs = 1000
└── StatLearn{SGD,LPDistLoss{2},L2Penalty}
    > β       : [0.925889, 1.72841, 3.14523, 3.86286, 4.66301]
    > λfactor : [0.1, 0.1, 0.1, 0.1, 0.1]
    > Loss    : L2DistLoss
    > Penalty : L2Penalty
    > Updater : SGD
```

4.4.8 Mean (`Mean`)

The mean of a random variable Y is the expected value $\mu = \mathbf{E}(y)$. The sample mean of a random sample from Y, $(y_1, \ldots, y_n)$, is defined as

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

The online update with arbitrary weight is

$$\theta^{(t)} = (1 - w_t)\theta^{(t-1)} + w_t y_t. \tag{4.50}$$

Mean Example

```
julia> Series(randn(1000), Mean())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── Mean(-0.00590063)
```

4.4.9 Non-Central Moments (`Moments`)

The `Moments` type estimates the first four non-central moments:

$$\mu'_k = \mathbf{E}(Y^k), \quad k = 1, 2, 3, 4. \tag{4.51}$$

The first four non-central sample moments of a random sample $(y_1, \ldots, y_n)$ are therefore

$$\hat{\mu}'_j = \frac{1}{n} \sum_{i=1}^{n} y_i^j, \quad j = 1, 2, 3, 4,$$

which can be adjusted for arbitrary weights as

$$\theta_j^{(t)} = (1 - w_t)\theta_j^{(t-1)} + w_t y_t^j, \quad j = 1, 2, 3, 4. \tag{4.52}$$

The skewness and kurtosis of Y are defined respectively as

$$\begin{aligned}\theta_1 &= \frac{E\left[(y-\mu)^3\right]}{E\left[(y-\mu)^2\right]^{3/2}} = \frac{\mu_3}{\sigma^3}, \\ \theta_2 &= \frac{E\left[(y-\mu)^4\right]}{E\left[(y-\mu)^2\right]^2} = \frac{\mu_4}{\sigma^4}.\end{aligned} \tag{4.53}$$

Skewness (measure of asymmetry) and kurtosis (measure of amount of curvature relative to a normal distribution) can use method-of-moments style online updates. While **OnlineStats** calculates non-central moments, Pébay (2008) provides formulas for singleton and minibatch updates to arbitrary order central moments.

Moments Example

```
julia> Series(randn(1000), Moments())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── Moments([0.0146969, 0.921228, 0.116298, 2.57592])
```

4.4.10 Order Statistics (`OrderStats`)

The order statistics $(y_{(1)}, \ldots, y_{(n)})$ of a random sample $(y_1, \ldots, y_n)$ are the rearranged values such that $y_{(1)} \leq y_{(2)} \leq \ldots \leq y_{(n)}$.

The `OrderStats` type is an estimator for the expected value of the order statistics of a finite sample of size b: $\mathbf{E}(y_{(1)}), \ldots \mathbf{E}(y_{(b)})$. The parameter updates once for every b observed values. Let $y_t = (y_{t1}, \ldots, y_{tb})$ be the next b observations, so that updates are

$$\theta^{(t)} = (1 - w_t)\theta^{(t-1)} + w_t \textsf{sort}(y_{t1}, y_{t2}, \ldots, y_{tb}). \tag{4.54}$$

Trivially, $\theta^{(t)}$ is unbiased for $\mathbf{E}(y_{(1)}), \ldots \mathbf{E}(y_{(b)})$, but the first and last order statistics are not biased for the minimum and maximum, respectively. It is also not in general true that the middle order statistic is unbiased for the median.

OrderStats Example

```
julia> Series(randn(1000), OrderStats(5))
```

```
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── OrderStats([-1.14608, -0.509083, 0.0145675, 0.504255, 1.12706])
```

4.4.11 Parametric Density Estimation

OnlineStats implements a number of online estimation procedures for parametric density estimation. The naming of types follows that of **Distributions.jl** by placing `Fit` in front of the type of distribution. For example, to fit a normal distribution to data, one would use the `FitNormal` type in **OnlineStats**. For parametric densities, the `value` function in **OnlineStats** operates similarly to the `params` function in **Distributions**, in that the distribution type can be created through "splatting": `Normal(value(FitNormal())...)`.

- **Beta Distribution** (`FitBeta`)

  For the Beta density

$$f(y|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} y^{\alpha-1}(1-y)^{\beta-1}, \quad y \in [0,1], \tag{4.55}$$

  the method of moments update using online mean and variance $\mu^{(t)}$ and $v^{(t)}$ is

$$\begin{aligned} \alpha^{(t)} &= \mu^{(t)} * \left[\frac{\mu^{(t)}(1-\mu^{(t)})}{v^{(t)}} - 1\right] \\ \beta^{(t)} &= (1-\mu^{(t)}) \left[\frac{\mu^{(t)}(1-\mu^{(t)})}{v^{(t)}} - 1\right]. \end{aligned} \tag{4.56}$$

  FitBeta Example

```
julia> Series(rand(1000), FitBeta())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── FitBeta((1.03431, 1.01411))
```

- **Categorical Distribution** (`FitCategorical`)

The `FitCategorical` type estimates a categorical density

$$P(Y = s) = p_s, \quad s \in S \tag{4.57}$$

where S is a finite set of possible values Y can take. `FitCategorical` is parameterized by the data type, which must be provided to the constructor.

FitCategorical Example

```
julia> Series(rand(Bool, 1000), FitCategorical(Bool))
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── FitCategorical{Bool}([0.475, 0.525])

julia> Series(rand(1:5, 1000), FitCategorical(Int))
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── FitCategorical{Int64}([0.196, 0.177, 0.2, 0.215, 0.212])
```

- **Cauchy Distribution** (`FitCauchy`)

  For the Cauchy density

$$f(y|\mu, \sigma) = \frac{1}{\pi\sigma\left[1 + \left(\frac{x-\mu}{\sigma}\right)^2\right]}, \quad y \in \mathbb{R}, \tag{4.58}$$

  OnlineStats uses the estimate

$$\begin{aligned} \mu^{(t)} &= q_{.5}^{(t)} \\ \sigma^{(t)} &= \frac{q_{.75}^{(t)} - q_{.25}^{(t)}}{2}, \end{aligned} \tag{4.59}$$

  where $q_{.25}^{(t)}, q_{.5}^{(t)}, q_{.75}^{(t)}$ are the current online estimates of the 0.25, 0.5, 0.75 quantiles.

FitCauchy Example

```
julia> using Distributions

julia> d = Cauchy(2,4)
Distributions.Cauchy{Float64}(μ=2.0, σ=4.0)

julia> Series(rand(d, 100_000), FitCauchy())
▩ Series{0} with LearningRate(r = 0.6)
  ├── nobs = 100000
  └── FitCauchy{SGD}((2.01174, 3.97727))
```

- **Gamma Distribution** (`FitGamma`)

  For the Gamma density

$$f(y|\alpha,\theta) = \frac{x^{\alpha-1}e^{-x/\theta}}{\Gamma(\alpha)\theta^{\alpha}}, \quad y > 0, \tag{4.60}$$

  the method of moments estimate based on online mean and variance $\mu^{(t)}$ and $v^{(t)}$ is

$$\begin{aligned} \theta^{(t)} &= \frac{v^{(t)}}{\mu^{(t)}}, \\ \alpha^{(t)} &= \frac{\mu^{(t)}}{\theta^{(t)}}. \end{aligned} \tag{4.61}$$

FitGamma Example

```
julia> using Distributions

julia> d = Gamma(5, 1)
Distributions.Gamma{Float64}(α=5.0, θ=1.0)

julia> Series(rand(d, 1000), FitGamma())
▩ Series{0} with EqualWeight
```

```
├── nobs = 1000
└── FitGamma((4.64513, 1.09999))
```

- **Log-Normal Distribution** (`FitLogNormal`)

  For the Log-Normal density

$$f(y|\mu,\sigma) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right), \quad y > 0, \tag{4.62}$$

  the maximum likelihood estimates for μ and σ^2 are the online mean and variance for $\ln y$:

$$\begin{aligned} \mu^{(t)} &= (1-\gamma_t)\mu^{(t-1)} + \gamma_t \ln y_t, \\ v^{(t)} &= (1-\gamma_t)v^{(t-1)} + (\ln y_t - \mu^{(t-1)})(\ln y_t - \mu^{(t)}). \end{aligned} \tag{4.63}$$

  FitLogNormal Example

```
julia> using Distributions

julia> d = LogNormal(10, 5)
Distributions.LogNormal{Float64}(μ=10.0, σ=5.0)

julia> Series(rand(d, 1000), FitLogNormal())
▩ Series{0} with EqualWeight
  ├── nobs = 1000
  └── FitLogNormal((9.99955, 5.30363))
```

- **Multinomial Distribution** (`FitMultinomial`)

  For the multinomial density

$$\frac{n!}{x_1!x_2!\ldots x_d!} p_1^{x_1} p_2^{x_2} \cdots p_d^{p_d}, \tag{4.64}$$

where each x_j is a positive integer of counts and $\sum p_j = 1$. In this case, each observation is $y_t = (x_{t1}, \ldots, x_{td})$, a vector of counts. The parameters of interest are $\theta = p_1, \ldots, p_d$ which are estimated with a multivariate mean $\mu = (\mu_1, \ldots, \mu_j)$:

$$\begin{aligned} \mu_j^{(t)} &= (1 - \gamma_t)\mu_j^{(t-1)} + \gamma_t x_{tj}, \quad j = 1, \ldots d, \\ \theta^{(t)} &= \left(\frac{1}{\sum_j \mu_j} \right) \mu. \end{aligned} \tag{4.65}$$

FitMultinomial Example

```
julia> d = Multinomial(1, [.2, .3, .5])
Distributions.Multinomial{Float64}(n=1, p=[0.2, 0.3, 0.5])

julia> Series(rand(d, 1000)', FitMultinomial(3))
▦ Series{1} with EqualWeight
  ├── nobs = 1000
  └── FitMultinomial((1, [0.186, 0.305, 0.509]))
```

- **Multivariate Normal Distribution** (`FitMvNormal`)

  For the multivariate normal density

$$f(y|\mu, \Sigma) \det(2\pi\Sigma)^{-1/2} \exp\left[-\frac{1}{2}(y - \mu)^T \Sigma^{-1} (y - \mu) \right], \tag{4.66}$$

  the maximum likelihood estimates for μ and Σ are the online mean and (biased) covariance matrix, respectively.

FitMvNormal Example

```
julia> Series(randn(1000, 2), FitMvNormal(2))
▦ Series{1} with EqualWeight
  ├── nobs = 1000
  └── FitMvNormal(([-0.0248385, -0.0239906],
   [0.976219 0.0277406; 0.0277406 1.06761]))
```

- **Normal Distribution** (`FitNormal`)

  For the Normal Density

$$f(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[\frac{(y-\mu)^2}{2\sigma^2}\right], \quad y \in \mathbb{R}, \tag{4.67}$$

  the maximum likelihood estimates for μ and σ^2 are the online mean and (biased) variance, respectively.

  FitNormal Example

```
julia> Series(randn(1000), FitNormal())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── FitNormal((0.0500511, 1.03526))
```

4.4.12 Quantiles (`Quantiles`)

Let Y be a random variable with cumulative distribution function $F_Y(y)$. The τ-th quantile ($0 < \tau < 1$) is defined as

$$Q_\tau(Y) = \inf\{y : F_Y(y) \geq \tau\}.$$

For example, the 0.5 quantile is more commonly known as the median. Define the quantile loss function (also known as check-loss) for the τ-th quantile as

$$\rho_\tau(u) = u\left[\tau - I(u < 0)\right]. \tag{4.68}$$

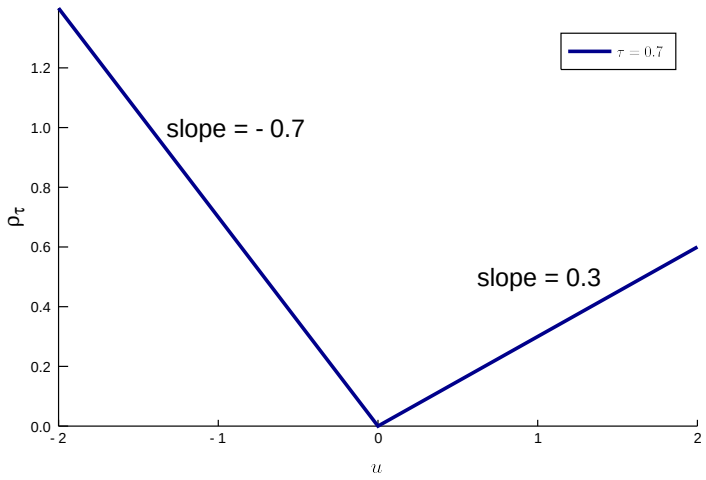


Figure 4.6: Quantile Loss for $\tau = .7$

The τ-th quantile of Y, $Q_\tau(Y)$, is known to be $\arg\min_\theta \mathbf{E}[\rho_\tau(Y-\theta)]$. Similarly, the τ-th sample quantile is

$$\arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \rho_\tau(y_i - \theta). \tag{4.69}$$

The sample quantile cannot be updated analytically, as quantile algorithms are based on order statistics (Hyndman and Fan, 1996). Since exact updates are not possible, `Quantile` is a subtype of `StochasticStat` and optionally allows the user to choose the stochastic approximation algorithm to be used.

SGD

`Quantile{SGD}` uses stochastic (sub)gradient descent. The quantile loss function is not differentiable, and therefore uses subgradients, although the only point the subgradient is not unique is at 0.

$$\theta^{(t)} = \theta^{(t-1)} - w_t[I(\theta^{(t-1)} < y_t) - \tau] \tag{4.70}$$

OMAS

`Quantile{OMAS}` uses a majorization of the quantile loss function discussed in Hunter and Lange (2004):

$$\begin{aligned} h_t(\theta) &= \frac{1}{4}\left[\frac{(y_t-\theta)^2}{v_t} + (4\tau-2)(y_t-\theta) + v_t\right] \\ &= \frac{1}{4}\left[\frac{\theta^2}{v_t} - (4\tau - 2 + 2y_t/v_t)\theta\right] + c_t, \end{aligned} \tag{4.71}$$

where $v_t = |y_t - \theta^{(t-1)}| + \epsilon$ for some small $\epsilon > 0$ and c_t contains the terms that do not depend on θ. The OMAS update can be derived as:

$$\begin{aligned} v_t &= \frac{1}{|y_t - \theta^{(t-1)}| + \epsilon} \\ s_1^{(t)} &= (1-w_t)s_1^{(t-1)} + w_t v_t y_t \\ s_2^{(t)} &= (1-w_t)s_2^{(t-1)} + w_t v_t \\ \theta^{(t)} &= \frac{s_1^{(t)} + 2\tau - 1}{s_2^{(t)}} \end{aligned} \tag{4.72}$$

MSPI

`Quantile{MSPI}` is based on the same majorization as `Quantile{OMAS}` and the update can be derived as

$$\begin{aligned} v_t &= \frac{1}{|y_t - \theta^{(t-1)}| + \epsilon} \\ \theta^{(t)} &= \frac{\theta^{(t-1)} + w_t(\tau - .5 + w_t y_t/2)}{1 + w_t v_t/2} \end{aligned} \tag{4.73}$$

Quantile Example

```
julia> q = [.25, .5, .75]
13-element Array{Float64,1}:
 0.25
 0.5
 0.75
```

```
julia> o1 = Quantile(q, SGD());

julia> o2 = Quantile(q, OMAS());

julia> 03 = Quantile(q, MSPI());

julia> Series(randn(100_000), o1, o2, o3)
▦ Series{0} with LearningRate(r = 0.6)
  ├── nobs = 100000
  ├── Quantile{SGD}([-0.68369, 0.00402352, 0.68859])
  ├── Quantile{OMAS}([-0.676623, 0.0109261, 0.681937])
  └── Quantile{MSPI}([-0.682194, 0.00377537, 0.687509])
```

4.4.13 Reservoir Sample (`ReservoirSample`)

A reservoir sample is a finite random sample without replacement of an unknown number of possible values. It is naturally an online algorithm, but it ignores any weighting mechanism. The update rule is:

$$
\begin{aligned}
j &= \begin{cases} t & \text{if } t \leq b \\ \text{random}(1, 2, \ldots, t) & \text{if } t > b \end{cases} \\
\theta_j^{(t)} &= \begin{cases} y_t & \text{if } j \leq b \\ \theta_j^{(t-1)} & \text{otherwise} \end{cases}.
\end{aligned}
\tag{4.74}
$$

Reservoir Sample Example

```
julia> Series(randn(1000), ReservoirSample(5))
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── ReservoirSample{([-1.183, -1.434, -0.144, -1.070, 1.394])
```

4.4.14 Sum (`Sum`)

The sum up to observation t is trivially calculated as

$$\theta^{(t)} = \theta^{(t-1)} + y_t. \tag{4.75}$$

`Sum` is parameterized by the data type to avoid loss of information, e.g. converting an `Int64` into a `Float64`.

Sum Example

```
julia> Series(1:100, Sum(Int))
▦ Series{0} with EqualWeight
  ├── nobs = 100
  └── Sum{Int64}(5050)

julia> Series(randn(1000), Sum(Float64))
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── Sum{Float64}(8.74307)
```

4.4.15 Variance (`Variance`)

The variance of a random variable Y is $\sigma^2 = \mathbf{E}[(Y - \mu)^2]$. The variance of a random sample $(y_1, \ldots, y_n)$ is defined as

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \mu)^2.$$

By dividing by $n - 1$ instead of n, the estimator is unbiased for the true value σ^2. We will henceforth refer to the *biased variance* estimator as the estimator which divides by n rather than $n - 1$.

The online algorithm for updating a variance analytically is given in Welford (1962), with generalizations to minibatch updates of more than one observation by Chan et al.

(1983). The arbitrary-weight update used by **OnlineStats** is

$$\begin{aligned}\mu^{(t)} &= (1 - w_t)\mu^{(t-1)} + w_t y_t, \\ \theta^{(t)} &= (1 - w_t)\theta^{(t-1)} + w_t(y_t - \mu^{(t-1)})(y_t - \mu^{(t)}).\end{aligned} \tag{4.76}$$

Variance Example

```
julia> Series(randn(1000), Variance())
▦ Series{0} with EqualWeight
  ├── nobs = 1000
  └── Variance(0.943157)
```

4.5 Extending OnlineStats

The basic interface of **OnlineStats** is defined by the lightweight **OnlineStatsBase** [9]. **Julia**'s use of multiple dispatch makes it easy to create new objects that work with **OnlineStats** with the same high performance as built-in types. Users are able to create their own `OnlineStat` and `Weight` types that work just as well as those that are available by loading the package.

4.5.1 User-Defined `OnlineStat`

An `OnlineStat` defined using methods from **OnlineStatsBase** is different than from **OnlineStats**. The interface follows similar function-naming conventions, but adds an underscore if necessary to avoid name conflicts with the imports in **OnlineStats**. The **OnlineStatsBase** interface is made up of `_fit!`, `_value!`, and `merge!`. The methods necessary for working with **OnlineStats** are described in section 4.2.1. Below is a simple example of creating an **OnlineStat** that counts the number of observations.

Counter Example

```
mutable struct Counter <: ExactStat{0}
    value::Int
```

[9] https://github.com/joshday/OnlineStatsBase.jl

```
    Counter() = new(0)
end
_value(o::Counter) = o.value  # this is unnecessary (default)
_fit!(o::Counter, y::Real, w::Float64) = (o.value += 1)
merge!(o::Counter, o2::Counter, w::Float64) = (o.value += o2.value)
```

4.5.2 User-Defined `Weight`

A `Weight` is simply a callable object with a single-argument method `myweight(n)`, where `n` is the current number of observations. An example of implementing a `Weight` via **OnlineStatsBase** is shown below.

ExponentialWeight Example

```
struct ExponentialWeight <: Weight
    λ::Float64
    ExponentialWeight(λ::Real = .1) = new(λ)
    ExponentialWeight(lookback::Integer) = new(2 / (lookback + 1))
end
(w::ExponentialWeight)(n) = n == 1 ? 1.0 : w.λ
```

4.6 Conclusion

In this chapter we introduced the **OnlineStats** package for calculating statistics with online algorithms. The scope of problems it solves is bigger than those of the existing software and it is easily extendible through **Julia**'s use of multiple dispatch. **OnlineStats** uses a novel unifying representation of online algorithms for statistics that allows new statistics and models to be implemented with a simple and straightforward interface. Since it is written in **Julia**, **OnlineStats** is not just fast, but uses the built-in parallelism features to allow algorithms to run online and in parallel.

CHAPTER

5

STOCHASTIC APPROXIMATION BEHAVIOR

5.1 Introduction

Stochastic approximation algorithms have received increased interest recently, as researchers seek scalable algorithms for optimizing objective functions. As they are often applied to machine learning problems, *stochastic approximation* and *online learning* are nearly synonymous terms. Traditional (offline) iterative algorithms for optimization such as Newton and quasi-Newton methods become prohibitively expensive as data gets larger. Therefore, new tools are needed for solving models applied to big data. The machinery behind stochastic approximation is not new (Robbins and Monro, 1951), but new algorithms based on this machinery are actively being developed.

It is difficult to compare stochastic approximation algorithms. In the online learning literature, theoretical error bounds are common, either with respect to the objective function $\ell(\theta)$ or to the euclidian distance between the current iterate and the true pa-

rameter ($\|\theta^{(t)} - \theta^*\|$). The theoretical guarantees of error bounds do little to inform the reader on how an algorithm behaves relative to other algorithms. Even if two algorithms have the same convergence rate, they may not agree on the set of assumptions and they may behave differently in practice. It is common to make comparisons on a benchmark dataset, such as the popular MNIST handwritten digits data. The drawbacks of a small number of dataset benchmarks are that it is impossible to extrapolate an algorithm's performance to different datasets, models, and selection of hyper-parameters. Another difficulty in the literature is that deep neural networks are often the model being fitted. Deep neural networks do not have a known solution, so algorithms cannot be evaluated on the merit of finding the "true" solution. Also, local minima often exist in neural network objectives so that different algorithms may simply be discovering different local minima.

Stochastic approximation algorithms involve a learning rate $\{\gamma_t\}_{t=1}^{\infty}$ that can have a dramatic effect on convergence. The learning rate is a positive sequence which approaches zero, intuitively creating a tradeoff between speed of convergence and variance of the iterates. However, this tradeoff does not occur uniformly across algorithms, and an optimal rate for one method may perform poorly with another. Also, theoretical bounds are often based on a specific choice of learning rate, leaving ambiguity in how the algorithm behaves under different (yet valid) rates. A standard assumption in stochastic approximation convergence theory is that the learning rate satisfies $\sum \gamma_t = \infty, \sum \gamma_t^2 < \infty$, and is often chosen to be

$$\gamma_t = t^{-r}, \quad r \in (0.5, 1]. \tag{5.1}$$

In practice $r = 0.5$ is also used even though it does not satisfy this assumption.

The goal of this chapter is to increase understanding of the behavior of online learning algorithms. We use Monte Carlo simulation to visualize the convergence of stochastic approximation algorithms over a variety of factors, such as type of model (with knowable solution) and learning rate hyper-parameter.

5.1.1 Simulations

The visualizations in the following sections demonstrate how algorithms are affected by different conditions. The factors studied are:

- **Type of Algorithm**. We use nine state-of-the-art stochastic approximation algorithms, described in the previous chapter.
- **Type of model**. We examine two objectives that are continuous (linear regression, logistic regression) and two that are not (distance weighted discrimination, quantile regression).
- **Learning rate hyper-parameter**. We use three learning rates parameterized as in (5.1) with $r \in \{.5, .7, .9\}$.

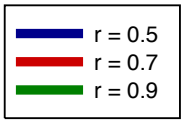


Figure 5.1: Legend for Learning Rate Parameters

- **Number of Parameters**. We also examine the difference between 10 and 50 predictor variables in each model.

For each combination of algorithm, model, learning rate hyper-parameter, and number of predictors, 100 replications are performed of:

1. **Simulating data**. In each case, the coefficient vector $\beta = (-1, \ldots, 1)$ is the linearly interpolated range from -1 to 1 and the independent variables are generated $x_t \overset{iid}{\sim} N(0, I_d)$ for an identity matrix I_d of appropriate size. The response variable y_t is generated in accordance with the model.
2. **Giving data to the learner**. For each observation in the simulated dataset, the estimate is updated ($\theta^{(t)}$) and then the offline objective value is recorded, relative to the minimum. We refer to this quantity as the the *relative loss*:

$$\frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta^{(t)}) - \min_{\theta} \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta). \tag{5.2}$$

For each replication, we record the relative loss after each update. In the following plots, the median relative loss across replications is plotted as a line with ribbons ranging from the 0.05 to 0.95 quantiles. Note that as lines approach 0, the online solution approaches the offline solution. The width of the ribbons therefore represents the variability of an algorithm's convergence under identical conditions. When the three bands are "close" to each other, this represents an algorithm's robustness towards the choice of learning rate.

The section for each statistical model begins with a description of the following:

- **Loss Function** (ℓ_t): The objective function of a single observation. This determines the form of the gradient for stochastic gradient algorithms.
- **Majorizing Function** (h_t): The majorizing function of a single observation (h_t majorizes ℓ_t at $\theta^{(t-1)}$). This determines the updates for majorization-based algorithms: OMAS, OMAP, and MSPI.
- **Data Model**: The process from which data is simulated.

Fair Comparison

In the interest of fairness, we omit several aspects of online learning that are typically included. A regularization function (i.e., ridge or LASSO) is usually added to avoid overfitting the model to data. Stochastic gradient algorithms can trivially include differentiable penalties such as ridge (L_2 norm), but some have not demonstrated how to incorporate a non-differentiable penalty such as LASSO (L_1 norm). To keep the algorithms on the same playing field, we omit regularization.

Since the introduction of stochastic gradient algorithms (Robbins and Monro, 1951), a large variety of techniques have been developed to speed up convergence or add stability. Some of these techniques are general enough to be used on a variety of stochastic gradient algorithms, such as the ideas of momentum (Rumelhart et al., 1988), Nesterov acceleration (Nesterov, 1983), and iterate averaging (Polyak and Juditsky, 1992). The most "fair" comparison across algorithms, is therefore a comparison without these techniques, since they can be applied separately to each algorithm.

Several of studied methods use hyper-parameters (beyond learning rate) that affect convergence. We choose the hyper-parameters as suggested by the paper that

introduced the method. Stochastic gradient algorithms typically multiply the learning rate by some positive constant. To keep learning rates the same across algorithms, we set this constant to 1.

5.2 Visualizations

5.2.1 Linear Regression

Loss Function

$$f_t(\beta) = \frac{1}{2}(y_t - x_t^T\beta)^2, \quad y_t \in \mathbb{R}. \tag{5.3}$$

Majorizing Function

$$h_t(\beta) = f_t(\beta^{(t-1)}) + \nabla f_t(\beta^{(t-1)})^T(\beta - \beta^{(t-1)}) + \frac{x_t^T x_t}{2}\|\beta - \beta^{(t-1)}\|^2. \tag{5.4}$$

Data Model

$$y_t = x_t^T\beta + \epsilon_t, \quad \epsilon_t \overset{iid}{\sim} N(0,1). \tag{5.5}$$

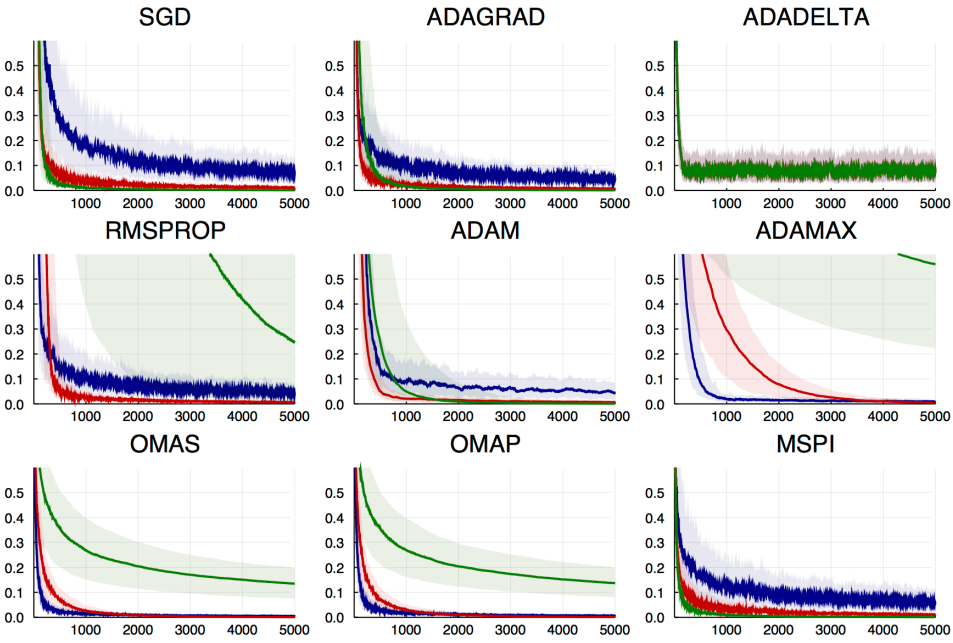


Figure 5.2: Linear Regression Simulation of 10 variables

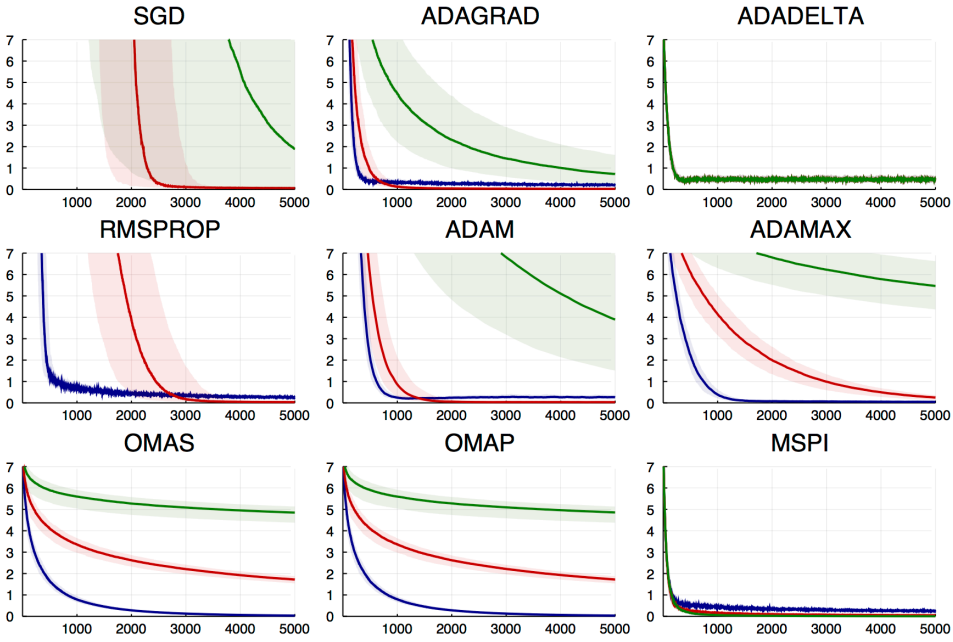


Figure 5.3: Linear Regression Simulation of 50 variables.

5.2.2 Logistic Regression

Loss Function

$$f_t(\beta) = \ln(1 + e^{y_t x_t^T \beta}), \quad y_t \in \{-1, 1\}. \tag{5.6}$$

Majorizing Function

$$h_t(\beta) = f_t(\beta^{(t-1)}) + \nabla f_t(\beta^{(t-1)})^T(\beta - \beta^{(t-1)}) + \frac{x_t^T x_t}{8}\|\beta - \beta^{(t-1)}\|^2. \tag{5.7}$$

Data Model

$$P(y_t = 1) = \frac{1}{1 + \exp(x_t^T \beta)}. \tag{5.8}$$

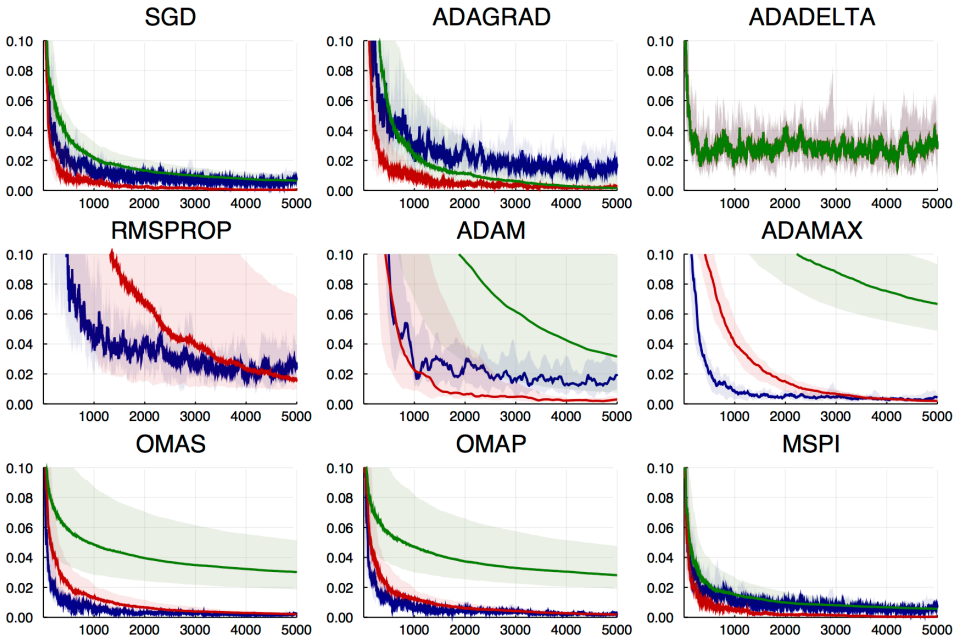


Figure 5.4: Logistic Regression Simulation of 10 variables

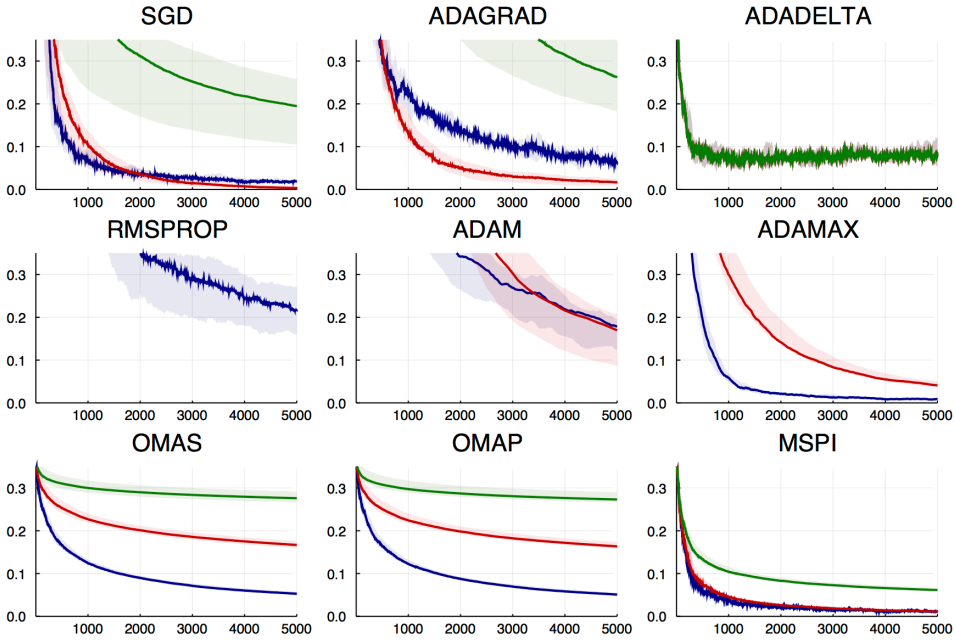


Figure 5.5: Logistic Regression Simulation of 50 variables

5.2.3 Distance Weighted Discrimination

The plots for Distance Weighted Discrimination differ from the other models. Rather than a different number of parameters, we use different values of the loss hyper-parameter q that demonstrates the adverse effect a larger q has on majorization-based learners.

Loss Function

$$f_t(\beta) = v_q(y_t x_t^T \beta), \quad y_t \in \{-1, 1\}, \tag{5.9}$$

where

$$\begin{aligned} v_q(u) &= \begin{cases} 1 - u & \text{if } u \leq \frac{q}{q+1} \\ \frac{q^q}{u^q (q+1)^{q+1}} & \text{if } u > \frac{q}{q+1}, \end{cases} \\ v_q'(u) &= \begin{cases} -1 & \text{if } u \leq \frac{q}{q+1} \\ \left[\frac{q}{u(q+1)}\right]^{q+1} & \text{if } u > \frac{q}{q+1}. \end{cases} \end{aligned} \tag{5.10}$$

Majorizing Function

$$h_t(\beta) = f_t(\beta^{(t-1)}) + \nabla f_t(\beta^{(t-1)})^T (\beta - \beta^{(t-1)}) + \frac{(q+t)^2 x_t^T x_t}{2q} \|\beta - \beta^{(t-1)}\|^2. \tag{5.11}$$

Data Model

$$P(y_t = 1) = \frac{1}{1 + \exp(x_t^T \beta)}. \tag{5.12}$$

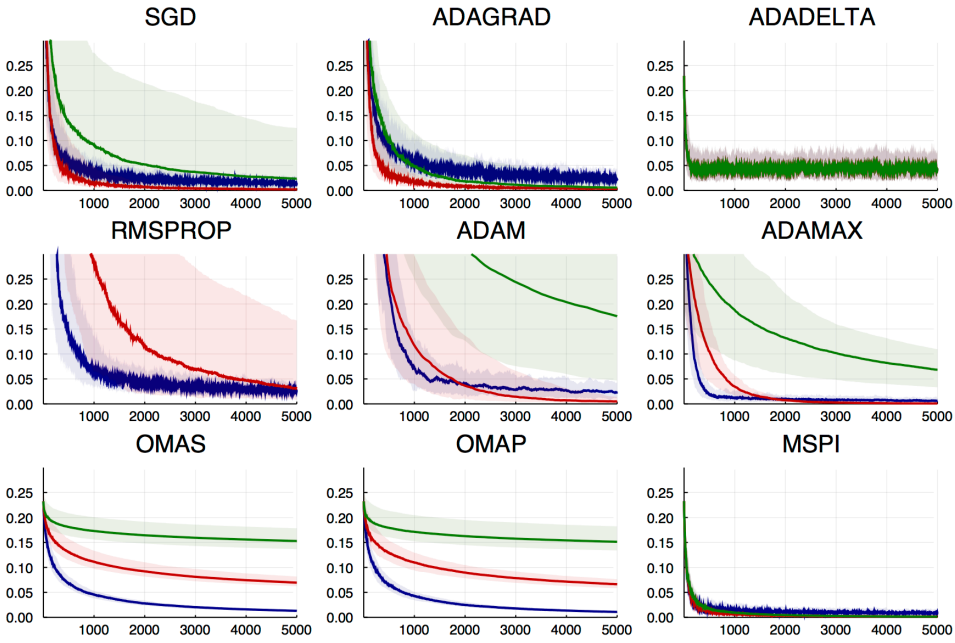


Figure 5.6: DWD Simulation for $q = .5$ of $d = 10$ variables

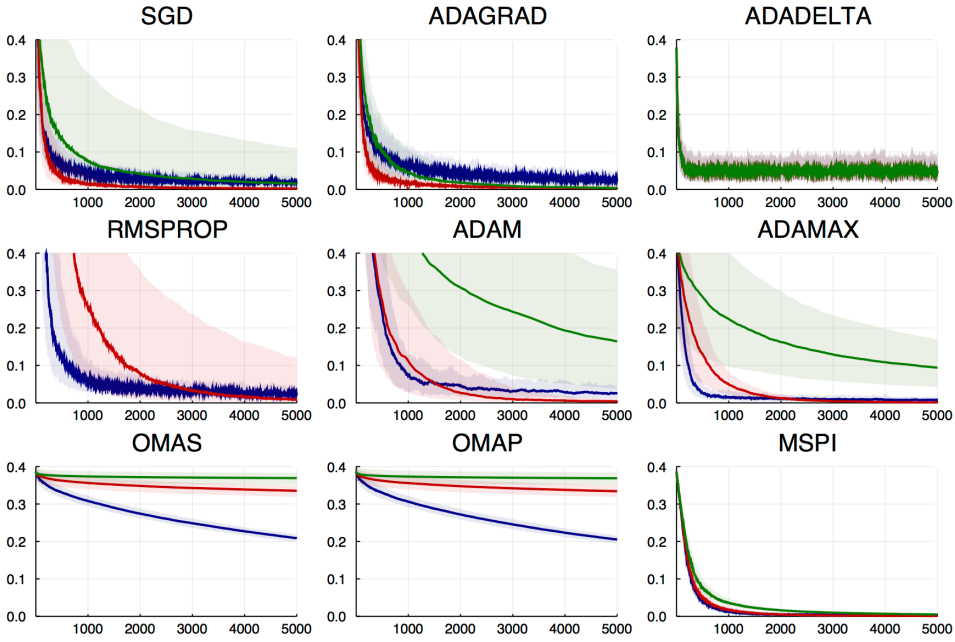


Figure 5.7: DWD Simulation for $q = 20$ of $d = 10$ variables

5.2.4 Quantile Regression

For quantile regression, OMAP is not possible with the majorization of choice, as the stochastic majorizations do not have a unique minimizer. Therefore, OMAP is omitted from the quantile regression plots.

Loss Function

$$f_t(\beta) = \rho_\tau(y_t - x_t^T\beta), \quad y_t \in \mathbb{R}. \tag{5.13}$$

where

$$\rho_\tau(u) = u(\tau - 1_{\{u<0\}}). \tag{5.14}$$

Majorizing Function

$$h_t(\beta) = \frac{1}{2}\frac{(x_t^T\beta)^2}{2w_t} - [y_t/(2w_t) + \tau - .5]x_t^T\beta + c_t, \tag{5.15}$$

where $w_t = \epsilon + |y_t - x_t^T\beta^{(t-1)}|$ and c_t contains terms which do not depend on β.

Data Model

$$y_t = x_t^T\beta + \epsilon_t, \quad \epsilon_t \overset{iid}{\sim} N(0,1). \tag{5.16}$$

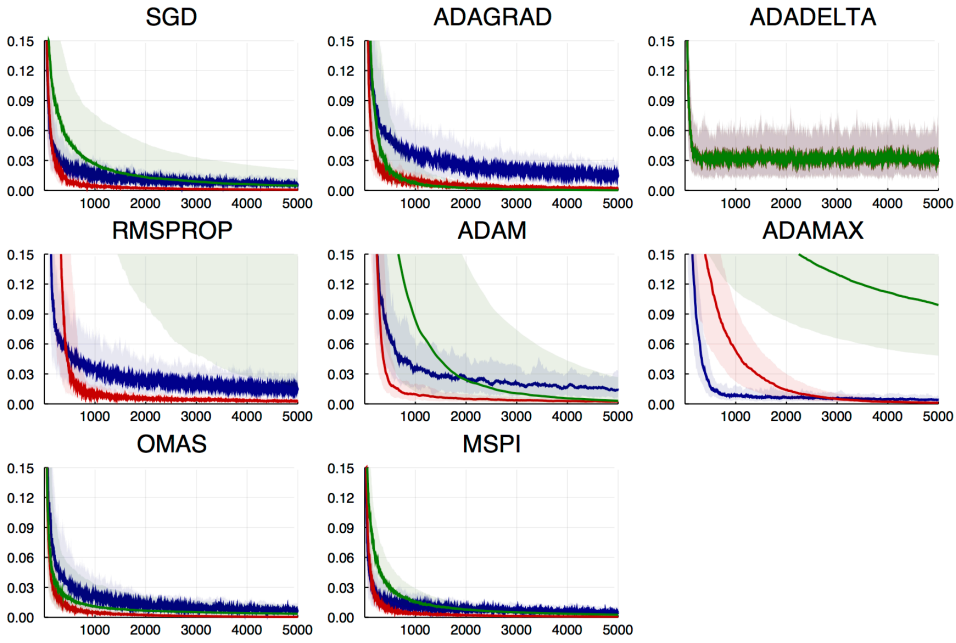


Figure 5.8: Quantile Regression Simulation with $\tau = 0.7$ of $d = 10$ variables

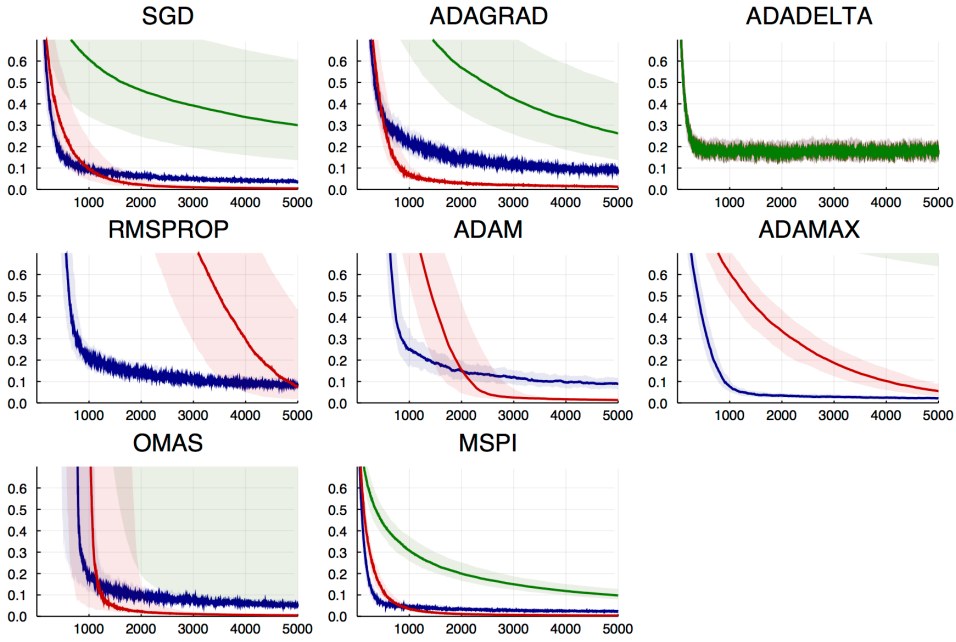


Figure 5.9: Quantile Regression Simulation with $\tau = 0.7$ of $d = 50$ variables

5.3 Conclusions

We presented visualizations of convergence behavior for several popular online learning algorithms alongside the algorithms based on majorizing functions (OMAS, OMAP, and MSPI). Intuitively, the latter are able to incorporate more information into each update, as some of the second order information of ℓ_t is contained in h_t. However, as seen with DWD, the majorization can slow down convergence if it is "too strongly convex". The cost of occasional slow convergence for online MM algorithms comes with the benefit of stability and low variability across learning rates.

The algorithms based on stochastic gradients (ADAGRAD, ADADELTA, RMSPROP, ADAM, ADAMAX) are generally considered upgrades to vanilla SGD. However, the best case learning rate for SGD coincides in nearly every model with the best case learning rate for the more complicated methods. Also, the worst case SGD often performs better than the others' worst cases. For example, looking only at worst cases for logistic regression, SGD converges faster and with less variability than the others for both 10 and 50 predictors. However, it may be that the modern developments are more robust to violated model assumptions and variance of the predictors, which would not be apparent in these simulations.

ADADELTA is somewhat different than the others as it does not use the learning rate, but has an *effective learning rate* which is based on the history of previous gradients. Even though it does not fully converge, ADADELTA very consistently appears to quickly converge to a neighborhood of the optimal value. A heuristic "search-then-converge" technique would be to begin learning with ADADELTA and then switch to another method for which step sizes shrink to zero.

Online MM algorithms (OMAS and OMAP) have very low variance in the error bounds, but convergence is heavily dependent on the learning rate. In every case, it appears a learning rate parameterized with $r = .5$ is the optimal choice. The two algorithms's behavior is nearly identical, and therefore in practice the easier of the two implementations should be used. As discussed in chapter 2, performing variable selection with regularization terms that set coefficients to zero are more effective with OMAS. Furthermore, as seen with quantile regression, OMAS can be applied even when the majorizing functions do not have a unique minimizer.

In every case, MSPI is a top contender in terms of both convergence rate and variability across all three learning rates. MSPI is stable, easy to implement, has the

same low computational effort as SGD, and can be applied to a wider class of problems than stochastic proximal iteration/implicit SGD. Using MSPI-Q, it is also straightforward to translate an existing (offline) MM algorithm via quadratic upper bound into an online algorithm.

Across all algorithms, there appears to be little benefit of a learning rate which rapidly approaches zero ($r = .9$). However, there is no obvious winner between $r = .5$ and $r = .7$ even when examining a specific method. Due to the sometimes drastic effect of learning rate choice, the method which is the most robust to a non-optimal rate should be preferred as a black box method being applied to a new problem. As the algorithm least sensitive to the learning rate, we recommend MSPI as the default choice in the online learning toolbox.

REFERENCES

Atchade, Y. F., Fort, G., and Moulines, E. (2014). On stochastic proximal gradient algorithms. *arXiv preprint arXiv:1402.2365*, 23.

Bach, F. R. and Moulines, E. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459.

Bates, D. (2013). Julia for R programmers. http://www.stat.wisc.edu/~bates/JuliaForRProgrammers.pdf.

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202.

Ben-Haim, Y. and Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11(Feb):849–872.

Bertsekas, D. P. (2011). Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98.

Blum, J. R. (1954). Approximation methods which converge with probability one. *The Annals of Mathematical Statistics*, pages 382–386.

Borkar, V. S. and Meyn, S. P. (2000). The ode method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469.

Bottou, L. (1998). Online learning and stochastic approximations. *On-line Learning in Neural Networks*, 17(9):142.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT 2010*, pages 177–186.

Bottou, L., Curtis, F. E., and Nocedal, J. (2017). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838v2*.

Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168.

Boyd, S. (2012). EE364b: Convex optimization II. http://web.stanford.edu/class/ee364b/index.html.

Boyd, S. (2014). EE364a: Convex optimization I. http://web.stanford.edu/class/ee364a/index.html.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122.

Cappé, O. and Moulines, E. (2009). On-line expectation-maximization algorithm for latent data models. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 71(3):593–613.

Casella, G. and Berger, R. L. (2002). *Statistical Inference*. Duxbury Pacific Grove, CA, 2 edition.

Chambers, J. M. (2017). *XRJulia: Structured Interface to 'Julia'*. R package version 0.7.6.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1983). Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247.

Chung, K. L. (1954). On a stochastic approximation method. *The Annals of Mathematical Statistics*, pages 463–483.

De Leeuw, J. and Lange, K. (2009). Sharp quadratic majorization in one dimension. *Computational Statistics & Data Analysis*, 53(7):2471–2484.

De Pierro, A. R. (1995). A modified expectation maximization algorithm for penalized likelihood estimation in emission tomography. *IEEE Transactions on Medical Imaging*, 14(1):132–137.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654.

Delyon, B., Lavielle, M., and Moulines, E. (1999). Convergence of a stochastic approximation version of the EM algorithm. *Annals of Statistics*, pages 94–128.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

Deng, H. and Wickham, H. (2011). Density estimation in r.

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Duchi, J. and Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10(Dec):2899–2934.

Fabian, V. (1968). On asymptotic normality in stochastic approximation. *The Annals of Mathematical Statistics*, pages 1327–1332.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics Springer, Berlin.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1.

Handian, D., Nasrulloh, I. F., and Riza, L. S. (2017). *gradDescent: Gradient Descent for Regression Tasks*. R package version 2.0.1.

Hunter, D. R. and Lange, K. (2000). Quantile regression via an mm algorithm. *Journal of Computational and Graphical Statistics*, 9(1):60–77.

Hunter, D. R. and Lange, K. (2004). A tutorial on mm algorithms. *The American Statistician*, 58(1):30–37.

Hyndman, R. J. and Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323.

Kaplan, A. and Tichatschke, R. (1998). Proximal point methods and nonconvex optimization. *Journal of Global Optimization*, 13(4):389–406.

Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466.

Kingma, D. and Ba, J. (2014). ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knuth, D. E. (1998). *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Pearson Education.

Koenker, R. (2005). *Quantile Regression*. Cambridge University Press.

Krakowski, K. A., Mahony, R. E., Williamson, R. C., and Warmuth, M. K. (2007). A geometric view of non-linear on-line stochastic gradient descent. Author webiste.

Kushner, H. and Yin, G. (2003). *Stochastic Approximation and Recursive Algorithms and Applications*. Stochastic Modelling and Applied Probability. Springer New York.

Lai, T. L. (2003). Stochastic approximation. *The Annals of Statistics*, 31(2):391–406.

Lange, K. (2010). *Numerical Analysis for Statisticians*. Springer Science & Business Media.

Lange, K. (2016). *MM Optimization Algorithms*. SIAM.

Lange, K., Hunter, D. R., and Yang, I. (2000). Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, 9(1):1–20.

Lumley, T. (2013). *biglm: bounded memory linear and generalized linear models*. R package version 0.9-1.

Mairal, J. (2013a). Optimization with first-order surrogate functions. In *ICML*, volume 3, pages 783–791.

Mairal, J. (2013b). Stochastic majorization-minimization algorithms for large-scale optimization. In *Advances in Neural Information Processing Systems*, pages 2283–2291.

Mairal, J. (2015). Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855.

McCullagh, P. (1984). Generalized linear models. *European Journal of Operational Research*, 16(3):285–292.

McLachlan, G. and Peel, D. (2004). *Finite Mixture Models*. John Wiley & Sons.

Monahan, J. F. (2008). *A Primer on Linear Models*. CRC Press.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376.

Nesterov, Y. (2009). Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1):221–259.

Nitanda, A. (2014). Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, pages 1574–1582.

Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239.

Pébay, P. (2008). Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical report, Sandia National Lab.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.

Qin, Z., Petricek, V., Karampatziakis, N., Li, L., and Langford, J. (2013). Efficient online bootstrapping for large scale learning. *arXiv preprint arXiv:1312.5021*.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407.

Robbins, H. and Siegmund, D. (1985). A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898.

Rosasco, L., Villa, S., and Vũ, B. C. (2014). Convergence of stochastic proximal gradient algorithm. *arXiv preprint arXiv:1403.5074*.

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747v2*.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1.

Ruppert, D. (1988). Efficient estimations from a slowly convergent Robbins-Monro process. Technical report, Cornell University Operations Research and Industrial Engineering.

Ryu, E. K. and Boyd, S. (2014). Stochastic proximal iteration: a non-asymptotic improvement upon stochastic gradient descent.

Ryzhov, I. O., Frazier, P. I., and Powell, W. B. (2011). A new optimal stepsize rule for approximate value iteration. Technical report, Princeton University. Working paper.

Sacks, J. (1958). Asymptotic distribution of stochastic approximation procedures. *The Annals of Mathematical Statistics*, 29(2):373–405.

Sakrison, D. J. (1965). Efficient recursive estimation; application to estimating the parameters of a covariance function. *International Journal of Engineering Science*, 3(4):461–483.

Schmidt, M., Le Roux, N., and Bach, F. (2017). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112.

Schraudolph, N. N., Yu, J., and Günter, S. (2007). A stochastic quasi-newton method for online convex optimization. In *Artificial Intelligence and Statistics*, pages 436–443.

Schäling, B. (2017). The boost c++ libraries. https://theboostcpplibraries.com.

Scott, D. W. (1985a). Averaged shifted histograms: Effective nonparametric density estimators in several dimensions. *The Annals of Statistics*, pages 1024–1040.

Scott, D. W. (1985b). Frequency polygons: theory and application. *Journal of the American Statistical Association*, 80(390):348–354.

Sherman, J. and Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127.

Spall, J. C. (2005). *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, volume 65. John Wiley & Sons.

Takane, Y., Young, F. W., and De Leeuw, J. (1977). Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. *Psychometrika*, 42(1):7–67.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31.

Titterington, D. M. (1984). Recursive parameter estimation using incomplete data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 257–267.

Toulis, P. and Airoldi, E. M. (2015). Implicit stochastic approximation. *arXiv preprint arXiv:1510.00967*.

Toulis, P., Airoldi, E. M., et al. (2017). Asymptotic and finite-sample properties of estimators based on stochastic gradients. *The Annals of Statistics*, 45(4):1694–1727.

Tran, D. and Toulis, P. (2016). *sgd: Stochastic Gradient Descent for Scalable Estimation*. R package version 1.1.

Wang, B. and Zou, H. (2015). Another look at dwd: Thrifty algorithm and bayes risk consistency in rkhs. *arXiv preprint arXiv:1508.05913*.

Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420.

Xiao, L. (2010). Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596.

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, Y. (2014). *Selected topics in statistical computing*. PhD thesis, NC State University.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *ICML*.