Chapter 8

Stochastic gradient / subgradient methods

Contents (class version)

8.0 Introduction

This chapter describes two families of algorithms:

- subgradient methods
- stochastic gradient methods aka stochastic gradient descent methods

Often we turn to these methods as a "last resort," for applications where none of the methods discussed previously are suitable. Many machine learning applications, such as training artificial neural networks, use such methods. As stated in [1] "large-scale machine learning represents a distinctive setting in which traditional nonlinear optimization techniques typically falter."

For recent surveys, especially about stochastic gradient methods, see [1, 2].

Acknowledgment

This chapter was based in part on slides made by Naveen Murthy in April 2019.

Rate of convergence review

Suppose the sequence $\{\boldsymbol{x}_k\}$ converges to $\hat{\boldsymbol{x}}$. Consider the limiting ratio

$$\mu \triangleq \lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\|_2}{\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2}.$$

We define the rate of convergence of the sequence $\{\boldsymbol{x}_k\}$ as follows:

- Converges *linearly* with rate μ if $\mu \in (0,1)$
- Converges *sublinearly* if $\mu = 1$
- Converges *super-linearly* if $\mu = 0$

Example: For $x_k = \rho^k$, with $|\rho| < 1$, $\dfrac{|x_{k+1} - 0|}{|x_k - 0|} = \dfrac{|\rho|^{k+1}}{|\rho|^k} = |\rho| = \mu$, so the sequence converges **linearly**.

Example: For $x_k = 1/k^c$, with $c > 0$, $\dfrac{|x_{k+1} - 0|}{|x_k - 0|} = \dfrac{1/(k+1)^c}{1/k^c} \to 0 = \mu$, so the convergence is **sublinear**.

For the sequence $x_k = 1 - \dfrac{1}{3^k} - \dfrac{5}{k^2}$, for $k = 1, 2, \ldots$, the value of μ is

A: 0 B: $1/5$ C: $1/3$ D: $1/2$ E: 1 **??**

Gradient descent (GD) review

- GD update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k).$$

- Converges to a minimizer if f is convex and **differentiable**, and ∇f is L-Lipschitz continuous, and $0 < \alpha < 2/L$; simple to analyze
- Worst-case **sublinear** rate of convergence of $\mathcal{O}(1/k)$ if $\alpha \leq 1/L$
- Can be improved to a linear rate $\mathcal{O}(\rho^k)$, where $\rho < 1$, under **strong convexity** assumptions on f
- In the usual case where $f(\boldsymbol{x}) = \sum_{m=1}^{M} f_m(\boldsymbol{x})$, gradient computation is *linear* in M, *i.e.*, takes $\mathcal{O}(M)$ time.
  $\Longrightarrow$ Doubling the number of examples in the training set doubles the gradient computation cost.
  GD is a "batch" method: ∇f uses all available data at once

The methods in this chapter relax the differentiability requirement, and scale better for large-scale problems.

Example. ImageNet [3] contains ∼14 million images with more than 20,000 categories.

8.1 The subgradient method

The $O(1/k)$ convergence rate of ISTA in Ch. 4 (aka **PGM**) may seem quite slow, and it is by modern standards, but convergence rates can be even worse! A classic way to seek a minimizer of a non-differentiable cost function Ψ is the **subgradient method**, defined as [4, 5]:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \qquad (8.1)$$

where $\boldsymbol{g}_k \in \partial \Psi(\boldsymbol{x}_k)$ denotes a **subgradient** of the (nondifferentiable) function Ψ at the current iterate $\boldsymbol{x}_k$.

This method was published (in Russian) by Naum Shor in 1962 for solving transportation problems [6, p. 4].

Subgradients and subdifferentials

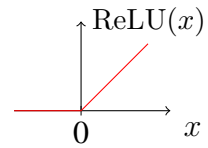
Define. If $f : \mathcal{D} \mapsto \mathbb{R}$ is a real-valued convex function defined on a **convex open set** $\mathcal{D} \subset \mathbb{R}^N$, a vector $\boldsymbol{g} \in \mathbb{R}^N$ is called a **subgradient** at a point $\boldsymbol{x}_0 \in \mathcal{D}$ iff for all $\boldsymbol{z} \in \mathcal{D}$ we have

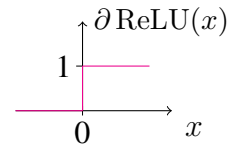
$$f(\boldsymbol{z}) - f(\boldsymbol{x}_0) \geq \langle \boldsymbol{g}, \boldsymbol{z} - \boldsymbol{x}_0 \rangle .$$

Define. The set of all subgradients at $\boldsymbol{x}_0$ is called the **subdifferential** at $\boldsymbol{x}_0$ and is denoted $\partial f(\boldsymbol{x}_0)$ [6, p. 8].

Example. A **rectified linear unit** (**ReLU**) in an ANN uses the rectifier function that has the following definition and **subdifferential**:

$$r(x) = \max(x, 0), \quad \partial r(x) = \begin{cases} 0, & x < 0 \\ , & x = 0 \\ 1, & 0 < x. \end{cases}$$





For this example, the derivative is defined **almost everywhere**, *i.e.*, everywhere but a set of **Lebesgue measure** equal to zero, also known as a **null set**. Specifically, here the derivative defined for the entire real line except for the point $\{0\}$. In most SIPML applications, the derivatives are defined except for a finite set of points.

Unfortunately, even for a convex function the direction negative to that of an arbitrary subgradient is not always a direction of descent [6, p. 4].

Example. For $f(x) = |x|$, the subdifferential is $\partial f(x) = \begin{cases} -1, & x < 0 \\ , & x = 0 \\ 1, & 0 < x. \end{cases}$

At $x = 0$ all elements of the subdifferential have negatives that are ascent directions except for 0.

Properties of subdifferentials

Define. The **subdifferential** of a **convex function** $f : \mathbb{R}^N \mapsto \mathbb{R}$ is this set of **subgradient** vectors:

$$\partial f(\boldsymbol{x}) \triangleq \left\{ \boldsymbol{g} \in \mathbb{R}^N \ : \ f(\boldsymbol{z}) - f(\boldsymbol{x}) \geq \langle \boldsymbol{g}, \boldsymbol{z} - \boldsymbol{x} \rangle, \ \forall \boldsymbol{z} \in \mathbb{R}^N \right\}.$$

Properties [7].

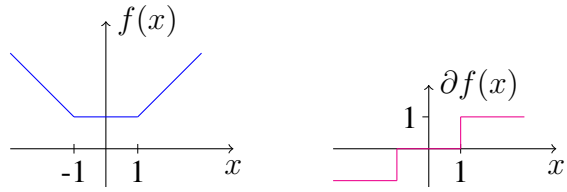
- A **convex function** $f : \mathcal{D} \mapsto \mathbb{R}$ is **differentiable** at $\boldsymbol{x} \in \mathcal{D}$ iff $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$.
  So a subgradient is a generalization of a gradient (for convex functions).
- For any $\boldsymbol{x} \in \mathcal{D}$, the subdifferential is a nonempty **convex** and **compact** set (closed and bounded) [6, p. 9].
- If convex functions $f, g : \mathbb{R}^N \mapsto \mathbb{R}$ have subdifferentials ∂f and ∂g, respectively, and $h(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$, then, for all $\boldsymbol{x} \in \mathbb{R}^N$, HW

$$\partial h(\boldsymbol{x}) = \partial(f+g)(\boldsymbol{x}) = \partial f(\boldsymbol{x}) + \partial g(\boldsymbol{x}) = \{\boldsymbol{u} + \boldsymbol{v} \ : \ \boldsymbol{u} \in \partial f(\boldsymbol{x}), \ \boldsymbol{v} \in \partial g(\boldsymbol{x})\},$$

  where the "+" here denotes the **Minkowski sum** of two sets.
  The subdifferential of a sum of convex functions is the (set) sum of their subdifferentials [6, p. 13].
- If convex function $f : \mathbb{R}^N \mapsto \mathbb{R}$ has subdifferential ∂f and $h(\boldsymbol{x}) \triangleq \alpha f(\boldsymbol{x})$ for $\alpha \in \mathbb{R}$, then $\partial h(\boldsymbol{x}) = \alpha \partial f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^N$. (?)
  A: True B: False **??**
  HW

- $\boldsymbol{x}$ is a **global minimizer** of a convex function f iff $\mathbf{0} \in \partial f(\boldsymbol{x})$ [6, p. 12].

  Example. $f(x) = \max(|x|, 1)$



- One can define convex functions and subdifferentials using the extended reals $\mathbb{R} \cup \{\infty\}$ [7].
- There are also generalization for non-convex functions [6, p. 17] [7].
- A **chain rule** for affine arguments [8]. If $g(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$ for convex $f : \mathbb{R}^M \mapsto \mathbb{R}$, for $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, then we saw in earlier HW that $g : \mathbb{R}^N \mapsto \mathbb{R}$ is convex, and furthermore [7, 8]:

$$\partial g(\boldsymbol{x}) = \boldsymbol{A}' \partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}). \tag{8.2}$$

  Proof that (Read)

$$\boldsymbol{v} \in \partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \Longrightarrow \boldsymbol{A}'\boldsymbol{v} \in \partial g(\boldsymbol{x}).$$

  Given that $\boldsymbol{v} \in \partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$, we know that $\forall \boldsymbol{y} \in \mathbb{R}^M$:

$$f(\boldsymbol{y}) - f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \geq \boldsymbol{v}' \left(\boldsymbol{y} - (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \right).$$

  In particular, choosing $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}$ for any $\boldsymbol{z} \in \mathbb{R}^N$ we have

$$\begin{aligned} f(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}) - f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) &\geq \boldsymbol{v}' \left((\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}) - (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \right) \\ &= \boldsymbol{v}' \boldsymbol{A} \left(\boldsymbol{z} - \boldsymbol{x} \right) = (\boldsymbol{A}'\boldsymbol{v})' \left(\boldsymbol{z} - \boldsymbol{x} \right). \end{aligned}$$

So by construction, for any $\boldsymbol{z} \in \mathbb{R}^N$ we have

$$g(\boldsymbol{z}) - g(\boldsymbol{x}) \geq (\boldsymbol{A}'\boldsymbol{v})' (\boldsymbol{z} - \boldsymbol{x})$$

implying that $\boldsymbol{A}'\boldsymbol{v} \in \partial g(\boldsymbol{x})$. Thus we have shown $\boldsymbol{A}'\partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \subseteq \partial g(\boldsymbol{x})$. □
Showing that $\partial g(\boldsymbol{x}) \subseteq \boldsymbol{A}'\partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$, to complete the proof of (8.2), seems to be more complicated.
For details: https://maunamn.wordpress.com/9-the-subdifferential-chain-rule

---

Example. Consider compressed sensing with analysis sparsity regularizer and $\beta \geq 0$:

$$\Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{T}\boldsymbol{x}\|_1 \Longrightarrow \boldsymbol{A}' (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + \beta \boldsymbol{T}' \operatorname{sign}.(\boldsymbol{T}\boldsymbol{x}) \in \partial \Psi(\boldsymbol{x}),$$

because $\operatorname{sign}(t)$ is a subgradient of $|t|$, where we have applied several of the above properties:

Convergence of the subgradient method

For suitable choice of $\{\alpha_k\}$, and suitable assumptions on Ψ such as convexity, the convergence rate of the subgradient method (8.1) is bounded by $O(1/\sqrt{k})$ [4–6]. (See result on subsequent pages.)

Diminishing step sizes

Convergence theorems for SGM often assume that the step sizes diminish, but not too quickly.
Specifically, often we assume they satisfy:

$$\alpha_k > 0, \quad \alpha_k \to 0, \quad \sum_{k=1}^{\infty} \alpha_k = \infty. \tag{8.3}$$

SGM convergence for diminishing step sizes

A classic convergence theorem for SGM is the following [6, p. 26].

If Ψ is convex and has a bounded set of minimizers $\mathcal{X}_*$, and $\{\alpha_k\}$ satisfies (8.3), then the sequence $\{\boldsymbol{x}_k\}$ generated by (8.4) for any $\boldsymbol{x}_0$ has the property that either

- the sequence $\{\boldsymbol{g}(\boldsymbol{x}_k)\}$ is bounded and the sequence $\{\boldsymbol{x}_k\}$ converges in the sense that $\{d(\boldsymbol{x}_k, \mathcal{X}_*)\} \to 0$ and $\{\Psi(\boldsymbol{x}_k)\} \to \Psi_*$, or
- the sequence $\{\boldsymbol{g}(\boldsymbol{x}_k)\}$ is unbounded and there is no convergence.

Note that this theorem does not ensure that $\{\boldsymbol{x}_k\}$ converges to some particular point $\hat{\boldsymbol{x}} \in \mathcal{X}_*$, only that it approaches the set. Of course if the set contains a single unique global minimizer, then the sequence converges to that point *if* it converges.

The "either" in the above theorem might seem unsatisfying, but often we know that the subgradients are bounded. In particular, if Ψ is piecewise linear with a finite number of pieces, then the subgradients are bounded (essentially by the maximum "slope").

The hinge loss function with 1-norm regularizer considered on p. 8.17 (and in HW) is piecewise linear with a finite number of pieces. (?)

A: True B: False **??**

Another theorem [6, p. 26] shows that if Ψ has a unique minimizer $\boldsymbol{x}_*$ and some other conditions hold, then $\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 \leq \alpha_{k+1}\sqrt{2}$, which implies $O(1/\sqrt{k})$ convergence due to (8.3).

Normalized subgradient method

Another way to avoid the "either" of the preceding convergence theorem is to use a **normalized subgradient method** of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \frac{\boldsymbol{g}_k}{\|\boldsymbol{g}_k\|_2}, \quad \boldsymbol{g}_k \in \partial \Psi(\boldsymbol{x}_k) \,. \tag{8.4}$$

Diminishing step size factors

This version has the following convergence theorem [6, p. 25] for diminishing step size factors that satisfy:

$$\eta_k > 0, \quad \eta_k \to 0, \quad \sum\nolimits_{k=1}^{\infty} \eta_k = \infty. \tag{8.5}$$

If Ψ is convex and has a bounded set of minimizers $\mathcal{X}_*$, and $\{\eta_k\}$ satisfies (8.5), then the sequence $\{\boldsymbol{x}_k\}$ generated by (8.4) for any $\boldsymbol{x}_0$ has the property that either $\boldsymbol{x}_K \in \mathcal{X}_*$ for some finite $K \in \mathbb{N}$, or $\{d(\boldsymbol{x}_k, \mathcal{X}_*)\} \to 0$ and $\{\Psi(\boldsymbol{x}_k)\} \to \Psi_*$.

Constant step-size factors

The normalized SGM (8.4) has the following convergence theorem for fixed step-size factors [4].

If Ψ is L-Lipschitz (itself, not its gradient!) then for $\eta_k = \epsilon / L$ [4, Thm. 1.2]:

$$T \geq \frac{L}{\epsilon} \|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2 \Longrightarrow \min_{k=0,\ldots,T} \Psi(\boldsymbol{x}_k) - \Psi_* \leq \epsilon.$$

It seems difficult to use this result in practice because we must choose ϵ in advance to specify η_k, and we must know $\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2$ to determine the number of iterations.

Projected subgradient method

For **constrained** optimization, *i.e.*, for minimizing Ψ subject to a convex constraint $\boldsymbol{x} \in \mathcal{C}$ where $\mathcal{C} \subset \mathbb{R}^N$, there is also an approach called the **projected subgradient** method:

$$\boldsymbol{x}_{k+1} = \mathcal{P}_{\mathcal{C}}\left(\boldsymbol{x}_k - \frac{\eta_k}{\|\boldsymbol{g}_k\|_2}\boldsymbol{g}_k\right), \quad \boldsymbol{g}_k \in \partial\,\Psi(\boldsymbol{x}_k)\,.$$

This algorithm has the following $O(1/\sqrt{k})$ convergence rate bound [5, Thm. 3, eqn. (21)].

If

- Ψ is L-Lipschitz on $\mathcal{C}$, *i.e.*, $\|\boldsymbol{g}\|_2 \le L,\ \forall \boldsymbol{g} \in \partial\,\Psi(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{C}$,
- $\Psi_* = \min_{\boldsymbol{x}\in\mathcal{C}}$ is finite,
- $\mathcal{X}_* = \{\boldsymbol{x} \in \mathcal{C} \ :\ \Psi(\boldsymbol{x}) = \Psi_*\}$ is nonempty,
- $R \triangleq \operatorname{dist}(\boldsymbol{x}_0, \mathcal{X}_*) = \|\boldsymbol{x}_0 - \mathcal{P}_{\mathcal{C}}(\boldsymbol{x}_0)\|_2\,,$
- and we choose $\eta_k = r/\sqrt{K+1}$ for some $r > 0$, and specified number of iterations K, then

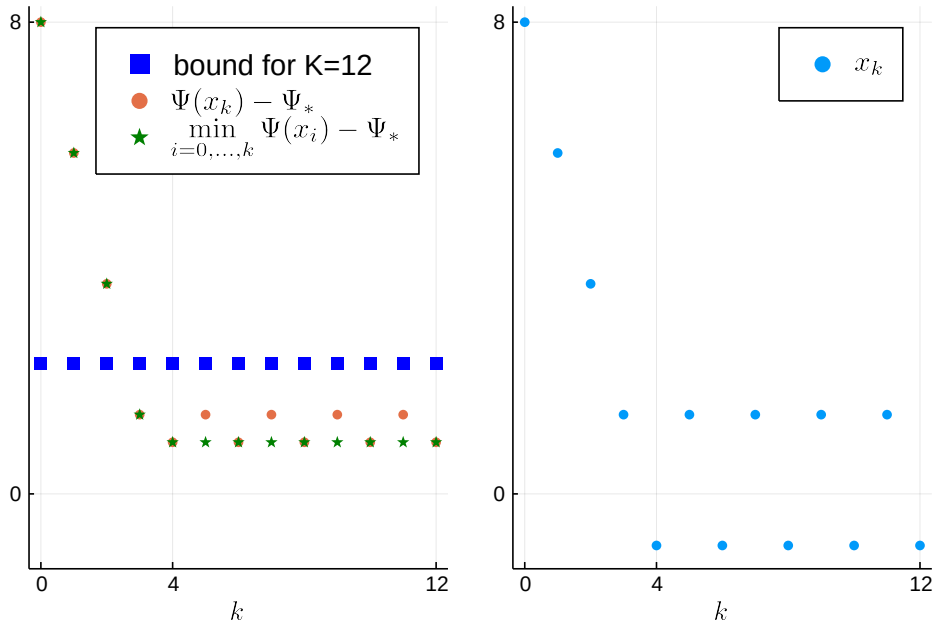
$$\min_{k=0,\ldots,K} \Psi(\boldsymbol{x}_i) - \Psi_* \le \frac{L}{2\sqrt{K+1}}\left(r + \frac{R^2}{r}\right).$$

What is the best choice of r for the step size?

A: R B: R^2 C: $1/R$ D: $1/R^2$ E: None **??**

The need to choose K in advance is unappealing.
Of course the ordinary normalized subgradient method is the special case where $\mathcal{C} = \mathbb{R}^N$.

Example. Consider the simple absolute value function $f(x) = |x|$ with $x_0 = 8$ and suppose we take the "oracle" choice $r = x_0$ and use the step size $\eta_k = r/\sqrt{K+1}$ for $K = 12$ iterations. The following figure shows how x_k evolves and how the cost function does indeed decrease to below the bound by iteration K. However, the sequence $\{x_k\}$ does not converge with this choice of step size and normalization.



Other notes about subgradient methods

There is a more general **proximal subgradient method** that has that same $O(1/\sqrt{k})$ rate [5].
This very slow convergence is why previous chapters described so many other algorithms.

Useful reference:
https://stanford.edu/class/ee364b/lectures/subgrad_method_notes.pdf

Here are additional references, some of which we may discuss if time permits:
[10] [11] [12] [13] [14] [15] [16]. For a survey, see [17].
A "conjugate subgradient" method is given in [18].

A "best" step-size rule of $1/\sqrt{k+1}$ is given in [19], where they state that a vanishing step size for subgradient method is "absolutely necessary."

8.2 Example: Hinge loss with 1-norm regularizer for binary classifier design

One motivating application is binary classifier design using the **hinge loss** function and a 1-norm to encourage parsimony for $M \times N$ matrix $\boldsymbol{A}$:

$$\Psi(\boldsymbol{x}) = \frac{1}{M} \mathbf{1}_M' h.(\boldsymbol{A}\boldsymbol{x}) + \beta \left\| \boldsymbol{x} \right\|_1, \quad h(t) = \max(1-t, 0).$$

Both terms in this cost function are non-smooth, but both are convex and have well defined subgradients.

For $g(x) = |x|$ the **subdifferential** is

$$\partial g(x) = \begin{cases} -1, & x < 0 \\ [-1, 1], & x = 0, \\ 1, & x > 0. \end{cases}$$

Thus for $g(\boldsymbol{x}) = \left\| \boldsymbol{x} \right\|_1$, an appropriate **subgradient** is

$$\operatorname{sign}.(\boldsymbol{x}) \in \partial g(\boldsymbol{x}).$$

For $h(t) = \max(1-t, 0)$ the **subdifferential** is

$$\partial h(t) = \begin{cases} -1, & t < 1, \\ [-1, 0], & t = 1 \\ 0, & t > 1. \end{cases}$$

Thus for $f(\boldsymbol{x}) = \mathbf{1}'h(\boldsymbol{A}\boldsymbol{x})$ an appropriate **subgradient** is

$$\boldsymbol{A}'\dot{h}.(\boldsymbol{A}\boldsymbol{x}) \in \partial f(\boldsymbol{x}), \text{ where } \dot{h}(t) \triangleq -\operatorname{sign}(1-t)\,.$$

Thus a **subgradient method** for solving this problem has an update like

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \left(\frac{1}{M} \boldsymbol{A}'\dot{h}.(\boldsymbol{A}\boldsymbol{x}_k) + \operatorname{sign}.(\boldsymbol{x}_k) \right),$$

for an appropriate step-size sequence $\{\alpha_k\}$.

A HW problem will compare the subgradient method to ADMM for this application.

Preview of stochastic subgradient method

To apply a **stochastic subgradient method** to this problem, we write the cost function as:

$$\Psi(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\boldsymbol{x}), \quad f_m(\boldsymbol{x}) \triangleq h([\boldsymbol{A}\boldsymbol{x}]_m) + M\beta \left\|\boldsymbol{x}\right\|_1 .$$

For each update we pick m at random, compute a **subgradient** $\boldsymbol{g}_m(\boldsymbol{x}_k)$ of f_m, select a step size α_k, and update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_m(\boldsymbol{x}_k)$. Then repeat until some convergence criterion is reached.

8.3 Incremental (sub)gradient method

Problem setup

Many applications in signal processing and machine learning involve finding the minimizer of a cost function $f : \mathbb{R}^N \to \mathbb{R}$ that is the sum (or average) of M functions:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \arg\min_{\boldsymbol{x}} \frac{1}{M} \sum_{m=1}^{M} f_m(\boldsymbol{x}). \tag{8.6}$$

- This form arises in many signal processing problems because sensor data often has independent noise, so the negative log-likelihood of the measurements becomes a summation.
- This form arises in many (supervised) machine learning problems because we assume that the training data consists of independent samples from some joint distribution and the average above is an empirical estimate of the risk or loss [1].

Example: Regularized **logistic regression**

Consider M training examples with feature vectors $\{\boldsymbol{v}_m\} \in \mathbb{R}^N$ and labels $y_m \in \{-1, +1\}$. The problem of learning weights $\boldsymbol{x} \in \mathbb{R}^N$ for binary classification (using logistic loss) can be written as

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \arg\min_{\boldsymbol{x}} \frac{1}{M} \sum_{m=1}^{M} f_m(\boldsymbol{x}), \quad f_m(\boldsymbol{x}) = h(\langle y_m \boldsymbol{v}_m, \boldsymbol{x} \rangle) + M\beta \left\| \boldsymbol{x} \right\|_1, \quad h(z) = \log\left(1 + \mathrm{e}^{-z}\right).$$

Lipschitz constant

If each f_m in (8.6) is differentiable, with an L_m-Lipschitz continuous gradient

$$\|\nabla f_m(\boldsymbol{x}) - \nabla f_m(\boldsymbol{z})\|_2 \leq L_m \|\boldsymbol{x} - \boldsymbol{z}\|_2, \quad \forall \boldsymbol{x}, \boldsymbol{z} \in \mathrm{dom}(f),$$

then a Lipschitz constant for ∇f is $L \triangleq \frac{1}{M} \sum_{m=1}^{M} L_m$, though this might not be the best Lipschitz constant.

GD

If a cost function of the form (8.6) is differentiable, then the GD update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - \alpha \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\boldsymbol{x}_k).$$

The subgradient method is similar.
Both of those methods require computing all the terms in the (possibly large) sum.

Incremental (sub)gradient method

For cost functions having the summation form (8.6), the main idea of **incremental gradient methods** (for differentiable cost functions) and **incremental subgradient methods** (for non-differentiable cases) is to use just one term in the summation to update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_{m_k}, \qquad \begin{array}{ll} \boldsymbol{g} = \nabla f_{m_k}(\boldsymbol{x}_k), & f_{m_k} \text{differentiable} \\ \boldsymbol{g} \in \partial f_{m_k}(\boldsymbol{x}_k), & \text{otherwise,} \end{array}$$

where here we choose the indexes sequentially, aka **cyclic ordering**: $m_k = 1 + k \mod M$.

This approach has a long history in the optimization literature; *e.g.*, the 1979 Russian paper [20] in turn cites a 1966 paper. Early work focused on the differentiable case whereas later work extended the analysis to subgradients [9, 17, 21–25]. There are also proximal versions [26] and versions based on majorization [27]. In the signal processing literature, a related approach is the **least-mean squares** (**LMS**) algorithm [2, 28].

A typical convergence result is [17, Prop. 3.4]. If the step sizes α_k satisfy (8.3), then $\liminf_{k\to\infty} f(\boldsymbol{x}_k) = f_*$. Furthermore if $\mathcal{X}_*$ is nonempty and $\sum_k \alpha_k^2 < \infty$, then $\{\boldsymbol{x}_k\}$ converges to some $\boldsymbol{x}_* \in \mathcal{X}_*$.

The theorem cited above actually covers more general cases where the cost function is composite with a combination of prox-friendly and non-differentiable terms needed a subgradient approach.

In machine learning, it is more common to pick the indices m at random, rather than cyclically, leading to the **stochastic (sub)gradient** method discussed next.

Keeping in mind that typically we prefer larger step sizes to achieve faster convergence, which of these step size sequences $\{\alpha_k\}$ satisfies the conditions of the previous convergence result and is most preferable?

A: 1 B: $1/\sqrt{k}$ C: $1/k$ D: $1/k^{3/2}$ E: $1/k^2$

$1/k$, the harmonic series, satisfies $\sum_k 1/k = \infty,\ \sum_k 1/k^2 < \infty$.

8.4 Stochastic gradient (SG) method

SG update

When each f_m is differentiable, the **stochastic gradient** (**SG**) update is nearly identical to the **incremental gradient** update:

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

except that here the index m_k is drawn *randomly* from the set $\{1, 2, \ldots, M\}$.

- This approach estimates gradient using only *one* f_m function.
- Computation of each update is very fast: $\mathcal{O}(1)$ instead of $\mathcal{O}(M)$
- Intuition: In expectation, the stochastic gradient equals the full gradient, *i.e.*, it is an unbiased estimate of the full gradient. Specifically, if j is a discrete random variable drawn randomly (uniformly) from the set $\{1, 2, \ldots, M\}$, then

$$\mathsf{E}[\nabla f_j(\boldsymbol{x})] = \sum_{m=1}^{M} \nabla f_m(\boldsymbol{x}) \, \mathsf{p}(j = m) = \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\boldsymbol{x}) = \nabla f(\boldsymbol{x}).$$

- The SG approach and its variants are used widely in machine learning software frameworks.
- The name **stochastic gradient descent** is misleading because it does not always descend in general! These notes will use the term **stochastic gradient** method instead.

Minibatching

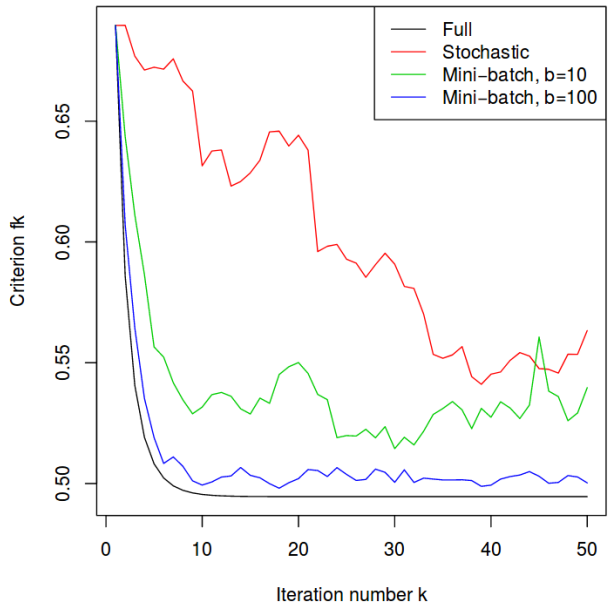
Instead of one data vector at a time, on can use groups of b data points called **mini batches**. The update becomes:

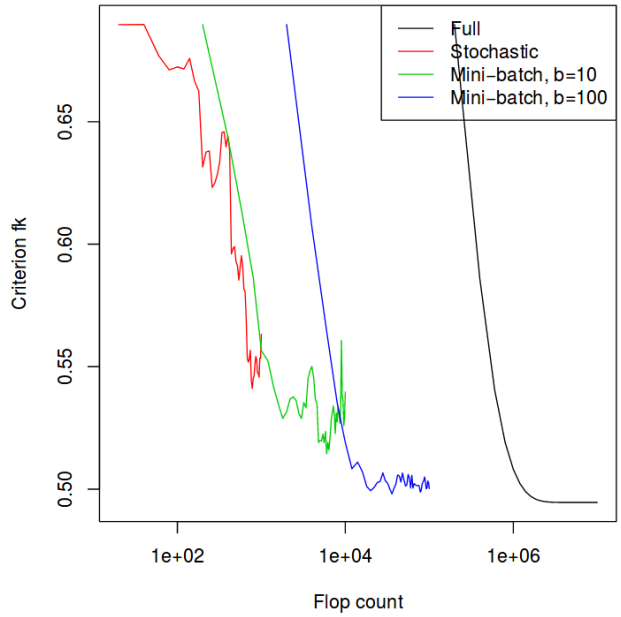
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \frac{1}{b} \sum_{m \in B_k} \nabla f_m(\boldsymbol{x}_k),$$

where B_k is drawn randomly from the set of all subsets of $\{1, 2, \ldots, M\}$ of size b.

- Intermediate approach between a stochastic gradient (SG) and the full gradient.
- Computational cost is more than SG $\mathcal{O}(b)$, but also reduces variance of the gradient estimate by a factor b.

Example: Regularized logistic regression. M = 10,000 and N = 20; fixed step-sizes used for all methods. (Slide adapted from Pradeep Ravikumar & Aarti Singh (CMU).)





Stochastic gradient algorithm: convergence analysis

Framework for analysis following [1]

Choose an initial iterate $\boldsymbol{x}_0$
for $k = 0, 1, \ldots$
- Generate a realization of random variable ξ_k
- Compute a stochastic vector $\boldsymbol{g}(\boldsymbol{x}_k, \xi_k)$
- Choose a step-size $\eta_k > 0$
- Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \boldsymbol{g}(\boldsymbol{x}_k, \xi_k)$

Here, $\boldsymbol{g}(\boldsymbol{x}_k, \xi_k)$ is a stochastic estimate of $\nabla f(\boldsymbol{x}_k)$.

Typically, ξ_k is simply m_k, the random index drawn from $\{1, \ldots, M\}$, and the stochastic vector $\boldsymbol{g}(\boldsymbol{x}_k, \xi_k)$ is simply $\nabla f(\boldsymbol{x}_k)$.

But the theory is presented more generally, to allow for mini-batches and other variations.

Stochastic gradients - bias & variance

Bias:

$$\text{bias}(g(\boldsymbol{x}_k, \xi_k)) \triangleq \mathsf{E}_{\xi_k}[g(\boldsymbol{x}_k, \xi_k)] - \nabla f(\boldsymbol{x}_k).$$

Variance:

$$\text{var}(g(\boldsymbol{x}_k, \xi_k)) \triangleq \mathsf{E}_{\xi_k}\left[\|g(\boldsymbol{x}_k, \xi_k)\|_2^2\right] - \|\mathsf{E}_{\xi_k}[g(\boldsymbol{x}_k, \xi_k)]\|_2^2 .$$

SG - fixed vs. diminishing step-sizes

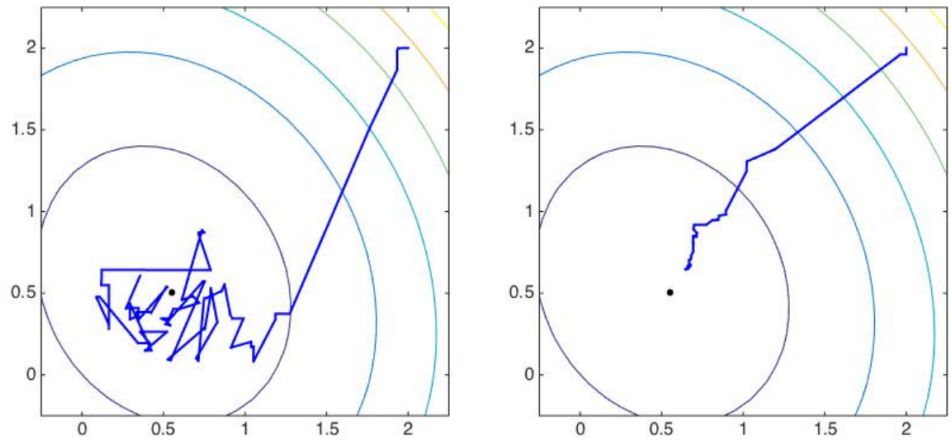


Figure: SG run with a fixed stepsize (left) vs. diminishing stepsizes (right)

Figure from F. Curtis, L. Bottou, J. Nocedal, "Beyond SG: Noise Reduction and Second-Order Methods," ICML, 2016.

Issues with fixed step-sizes:

- For fixed step-size, SG sequence approaches a noise ball (proportional to step-size) around the optimum.
- Choosing a lower constant size reduces the size of the noise ball, but leads to *slower convergence*.
- Intuition: use larger step-sizes initially and gradually decrease

SG: diminishing step-sizes

- Diminishing step-size schemes are often chosen to be $\mathcal{O}(1/k)$.

  For example (https://scikit-learn.org/stable/modules/sgd.html):

$$\eta_k = \frac{1}{a(k_0 + k)}$$

- The hyperparameters a and k_0 could be hand-tuned or chosen through an automatic process.
- Sufficient conditions for convergence:

$$\sum_{k=1}^{\infty} \eta_k = \infty, \qquad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

  So the step sizes should diminish, but not too quickly.

Convergence rates

- **Full (batch) GD:**
  For convex f, with L-Lipschitz gradient, under suitable stepsizes:

$$f(\boldsymbol{x}_k) - f_* = \mathcal{O}(1/k)$$

- **Stochastic gradient method:**
  For convex f, under diminishing stepsizes (along with other conditions):

$$\mathsf{E}[f(\boldsymbol{x}_k)] - f_* = \mathcal{O}(1/\sqrt{k}).$$

  (Here k can be a full iteration or a batch, the factor M is just a constant.)

Convergence rates (strong convexity)

Under **strong convexity** assumptions on f (with parameter μ)

- **Full GD:**
  For *strongly convex* f, With L-Lipschitz gradient, for suitable stepsizes GD has a **linear** rate:

$$f(\boldsymbol{x}_k) - f_* = \mathcal{O}(\rho^k), \text{ where } \rho < 1.$$

- **SG method:** Under strong convexity (plus other assumptions as before), SG sequence has **sublinear** rate:

$$\mathsf{E}[f(\boldsymbol{x}_k)] - f_* = \mathcal{O}(1/k)$$

  Can we do better than **sublinear** convergence for the SG method?

Improving the SG method

- Diminishing stepsizes (learning rates) are needed due to the variability of stochastic gradients.
- Can we reduce the variability of those stochastic gradients and use a constant learning rate?
- Can we use better learning rate schedules to accelerate SG method?

Variance reduction methods:

- SAG, SAGA [36],
- SVRG [37]
- S2GD, SDCA etc.
- SARAH [38]

Adjusting learning rate:

- Momentum, Nesterov accelerated gradient [39]
- **Adagrad**, **ADAM**, (**RMSProp**) etc. http://cs231n.github.io/neural-networks-3/
- Fixed/adaptive restart (**SGDR**) [40]

SVRG (Stochastic Variance Reduced Gradient)

Algorithm SVRG

Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0$, and positive integer m.
for $k = 1, 2, \ldots$ **do**
    Compute the batch gradient $\nabla F(w_k)$.
    Initialize $w_{k,1} \leftarrow w_k$.
    for $j = 1, \ldots, m$ **do**
        Chose i uniformly from $\{1, \ldots, n\}$.
        Set $g_{k,j} \leftarrow \nabla f_i(w_{k,j}) - (\nabla f_i(w_k) - \nabla F(w_k))$.
        Set $w_{k,j+1} \leftarrow w_{k,j} - \alpha g_{k,j}$.
    end for
    Option (a): Set $w_{k+1} = \tilde{w}_{m+1}$
    Option (b): Set $w_{k+1} = \frac{1}{m} \sum_{j=1}^{m} \tilde{w}_{j+1}$
    Option (c): Choose j uniformly from $\{1, \ldots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.
end for

Credit: Leon Bottou, Frank Curtis, Jorge Nocedal

SAGA (Stochastic Averaged Gradient)

Algorithm SAGA

Choose an initial iterate $w_1 \in \mathbb{R}^d$ and stepsize $\alpha > 0$.
for $i = 1, \ldots, n$ **do**
    Compute $\nabla f_i(w_1)$.
    Store $\nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1)$.
end for
for $k = 1, 2, \ldots$ **do**
    Choose j uniformly in $\{1, \ldots, n\}$.
    Compute $\nabla f_j(w_k)$.
    Set $g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]})$.
    Store $\nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k)$.
    Set $w_{k+1} \leftarrow w_k - \alpha g_k$.
end for

Credit: Leon Bottou, Frank Curtis, Jorge Nocedal

Variance reduction: overview

Consider 2 random variables X, Y. Define a new random variable as follows:

$$Z_\alpha \triangleq \alpha(X - Y) + \mathsf{E}[Y], \qquad \alpha \in [0, 1].$$

Can we design Z_α to obtain a "good" estimator of $\mathsf{E}[X]$?

- $\alpha = 1$ results in an unbiased estimator: $\mathsf{E}[Z_1] = \mathsf{E}[X]$.
- $\mathsf{Var}\{Z_\alpha\} = \alpha^2 \left(\mathsf{Var}\{X\} + \mathsf{Var}\{Y\} - 2\ \mathsf{Cov}\{X, Y\}\right)$.
- If X and Y are highly correlated, Z_α has reduced variance.

SG variants with reduced variance

Let X be the current SG direction $\nabla f_j(\boldsymbol{x}_k)$ and Y be a past stored gradient $\nabla f_j(\boldsymbol{\phi}_j^{(k)})$.

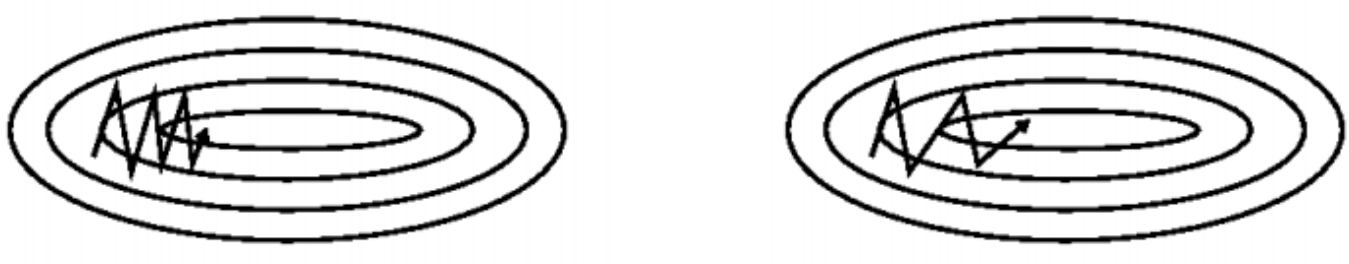
$$\text{(SAG)} \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \left(\frac{1}{M} \left(\nabla f_j(\boldsymbol{x}_k) - \nabla f_j(\boldsymbol{\phi}_j^{(k)}) \right) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\boldsymbol{\phi}_m^{(k)}) \right)$$

$$\text{(SAGA)} \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \left(\nabla f_j(\boldsymbol{x}_k) - \nabla f_j(\boldsymbol{\phi}_j^{(k)}) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\boldsymbol{\phi}_m^{(k)}) \right)$$

$$\text{(SVRG)} \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \left(\nabla f_j(\boldsymbol{x}_k) - \nabla f_j(\tilde{\boldsymbol{x}}) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\tilde{\boldsymbol{x}}) \right)$$

- (**SAG**) - Stochastic Average Gradient [41]
- (**SVRG**) - Stochastic Variance Reduced Gradient

Momentum



(a) SGD without momentum (b) SGD with momentum

$$\boldsymbol{v}_{k+1} = \mu \boldsymbol{v}_k - \eta_k \nabla f(\boldsymbol{x}_k)$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_{k+1}$$

- $\mu \in [0, 1]$ is the momentum coefficient; typically chosen to be high ($\sim$0.9)
- $\mu = 0$ same as ordinary GD

Image taken from [39]

Nesterov momentum

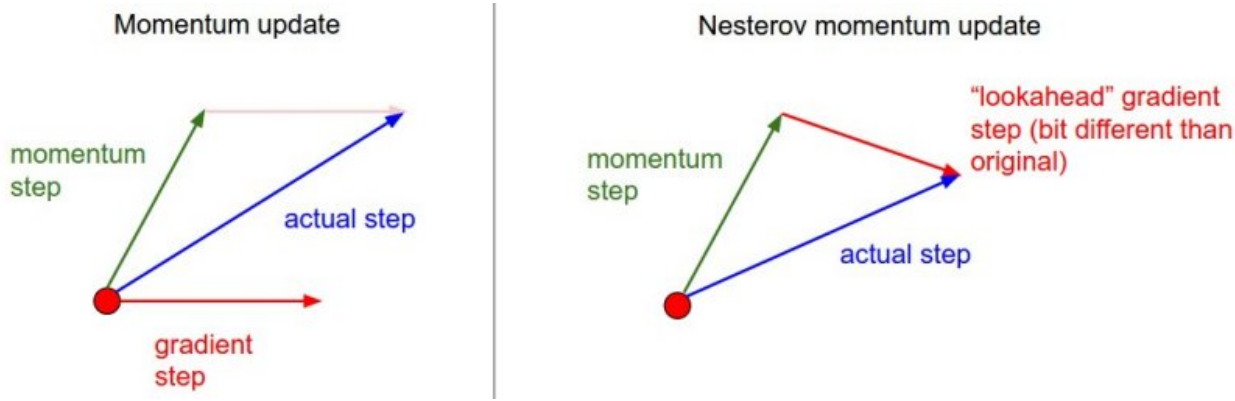


Image from http://cs231n.github.io/assets/nn3/nesterov.jpeg

$$\boldsymbol{v}_{k+1} = \mu \boldsymbol{v}_k - \eta_k \nabla f(\boldsymbol{x}_k + \mu \boldsymbol{v}_k)$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_{k+1}$$

- Gradient is computed at lookahead point $\boldsymbol{x}_k + \mu \boldsymbol{v}_k$ instead of at $\boldsymbol{x}_k$.
- $\mu = 0$ reverts to ordinary GD

Adaptive step-sizes

- Idea: Instead of assigning the same learning rate for each feature, why not vary the rate per feature (depending on importance)?
- Choose step-sizes adaptively based on measurements and heuristics (instead of an *a priori* schedule).
- Examples:

- **Adagrad**:
  - Tracks sum of squared gradients.
  - Learning rate is different for different directions (depending on past gradients).
  - Caveat: The learning rate decays too aggressively (due to monotonicity).
- **RMSprop**:
  - Similar to Adagrad, but looks at past gradients only over a moving window.
  - Due to this short-term memory, learning rates do not get monotonically smaller; better than Adagrad.
- **Adam**:
  - Can be thought of as RMSprop with momentum.
  - Includes bias correction terms for first and second moments.
  - Very widely used for training deep networks.

Optimizer choice

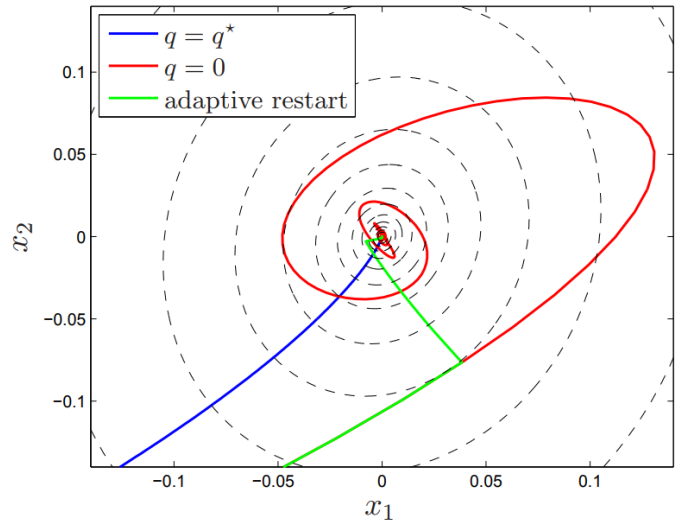
- No one-size-fits-all choice
- Use momentum approaches for higher training speed
- If dataset is sparse, adaptive learning rates are a reasonable choice
- Default algorithms to try:
  - Adam is widely used for training networks
  - SG + momentum is also surprisingly effective for many applications

Visualizations of optimizer trajectories:

- https://github.com/Jaewan-Yun/optimizer-visualization
- http://ruder.io/optimizing-gradient-descent/
- http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/

Restart

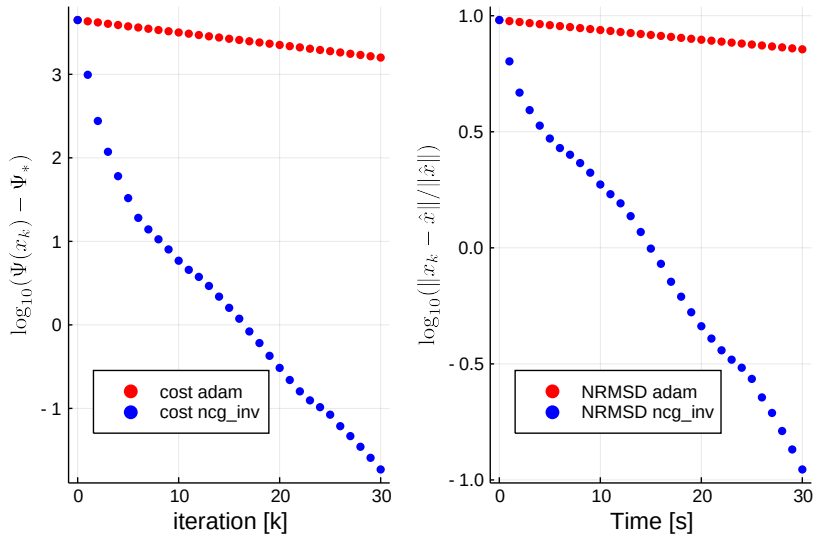
Blue - Optimal momentum (with strong convexity parameter known)
Red - Nesterov Momentum
Green - Adaptive restart
Image from [42]



- Function scheme: Restart whenever objective function value increases
- Gradient scheme: Restart when negative gradient and momentum form an obtuse angle
- Stochastic Gradient Descent with Warm Restarts (SGDR)

Example: Ordinary Least-Squares

This figure compares the ADAM optimizer (in the `Flux.jl` toolbox), using its default learning rate, to ordinary CG for an ordinary LS problem $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$. The paper on Adam [49] claims "The hyper-parameters have intuitive interpretations and typically require little tuning."
CG has no tuning parameters. The point here is that one should use the right tool for the job.



8.5 Example: X-ray CT reconstruction

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \Psi(\boldsymbol{x}) \triangleq \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_{\boldsymbol{W}}^2 + \beta R(\boldsymbol{x})$$

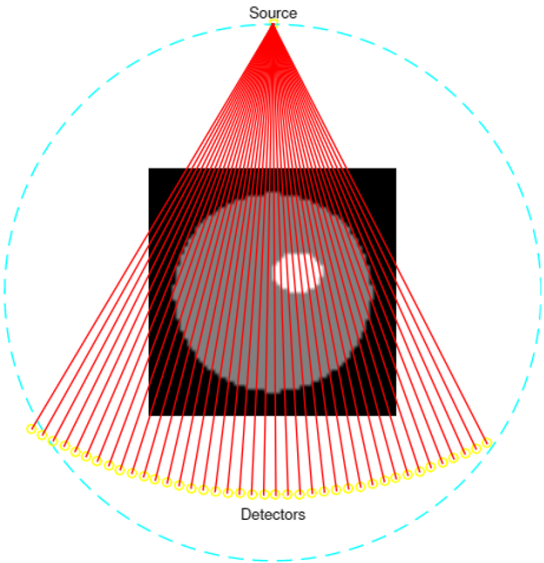
TABLE I
SQS METHODS

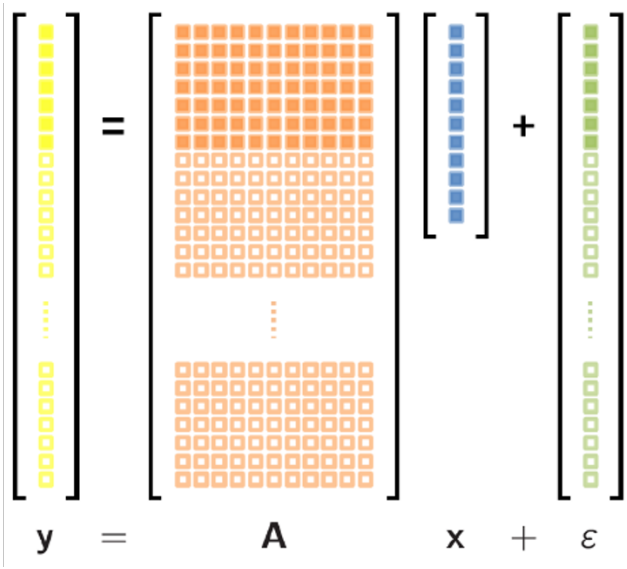
1: Initialize $\boldsymbol{x}^{(0)}$ and compute $\boldsymbol{D}$ such that (5) and (6) hold.
2: for $n = 0, 1, \cdots$
3: $\quad \boldsymbol{x}^{(n+1)} = \mathcal{P}_{\mathcal{X}} \left[\boldsymbol{x}^{(n)} - \boldsymbol{D}^{-1} \nabla \Psi(\boldsymbol{x}^{(n)}) \right]$

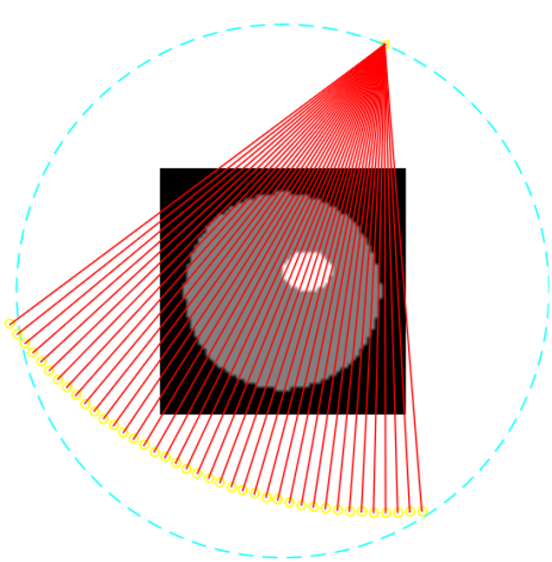
Can use an MM algorithm with $\boldsymbol{D} = \text{Diag}\{\boldsymbol{A}'\boldsymbol{W}\boldsymbol{A}\mathbf{1}\}$, called separable quadratic surrogates (SQS [50]

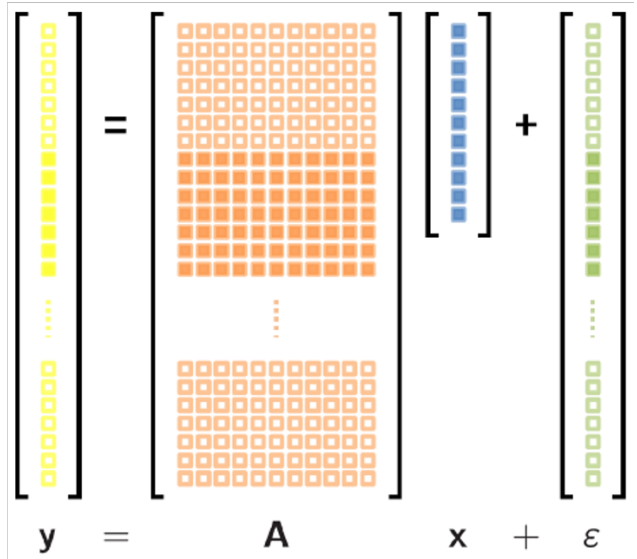
Background: X-ray CT model

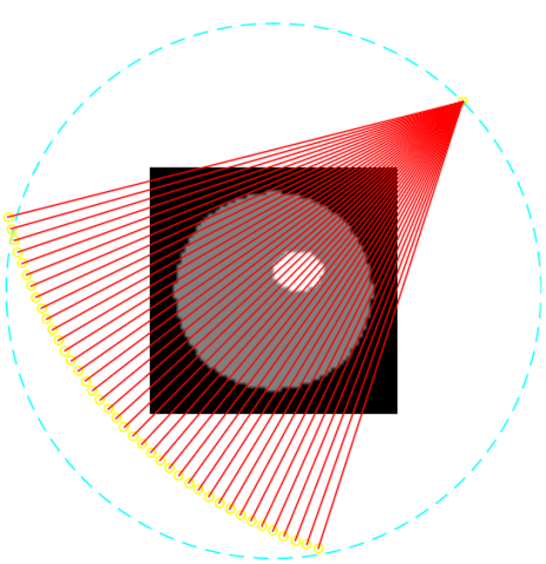
$\boldsymbol{x}$: Unknown attenuation image to be reconstructed
$\boldsymbol{y}$: Noisy sinogram data
$\boldsymbol{A}$: CT system matrix

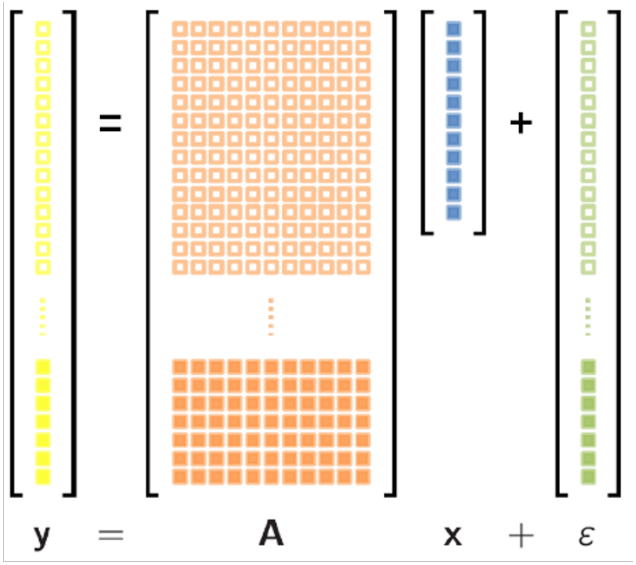

Source
Detectors


y = A x + ε




y = A x + ε




=
+
y
=
A
x
+
ε

Ordered subsets

- We can rewrite the cost function as:

$$\Psi(\boldsymbol{x}) = \sum_{m=1}^{M} \Psi_m(\boldsymbol{x}), \quad \Psi_m(\boldsymbol{x}) \triangleq \frac{1}{2} \|\boldsymbol{y}_m - \boldsymbol{A}_m \boldsymbol{x}\|_{\boldsymbol{W}_m}^2 + \frac{1}{M} R(\boldsymbol{x}), \quad m = 1, \ldots, M.$$

- Intuition:

  Divide the total projections into M "subsets" (akin to minibatches in a machine learning problem). The idea is to save computation by replacing a full gradient with a subset-specific gradient.

Ordered subsets + momentum

- Incorporate momentum into the algorithm with ordered subsets [50] (somewhat analogous to minibatch-SGD plus momentum).
- Choose subsets to maintain the subset balance approximation (to reduce variance):

$$\nabla \Psi(\boldsymbol{x}) \approx M \nabla \Psi_1(\boldsymbol{x}) \approx \ldots \approx M \nabla \Psi_M(\boldsymbol{x})$$

TABLE III

PROPOSED OS-SQS METHODS WITH MOMENTUM IN [17] (OS-MOM1)

Initialize $\boldsymbol{x}^{(0)} = \boldsymbol{z}^{(0)}$, $t_0 = 1$ and compute $\boldsymbol{D}$.

for $n = 0, 1, \cdots$

for $m = 0, 1, \cdots, M-1$

$\quad k = nM + m$

$\quad t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$

$\quad \boldsymbol{x}^{(\frac{k+1}{M})} = \mathcal{P}_{\mathcal{X}}\left[\boldsymbol{z}^{(\frac{k}{M})} - \boldsymbol{D}^{-1} M \nabla \Psi_m(\boldsymbol{z}^{(\frac{k}{M})})\right]$

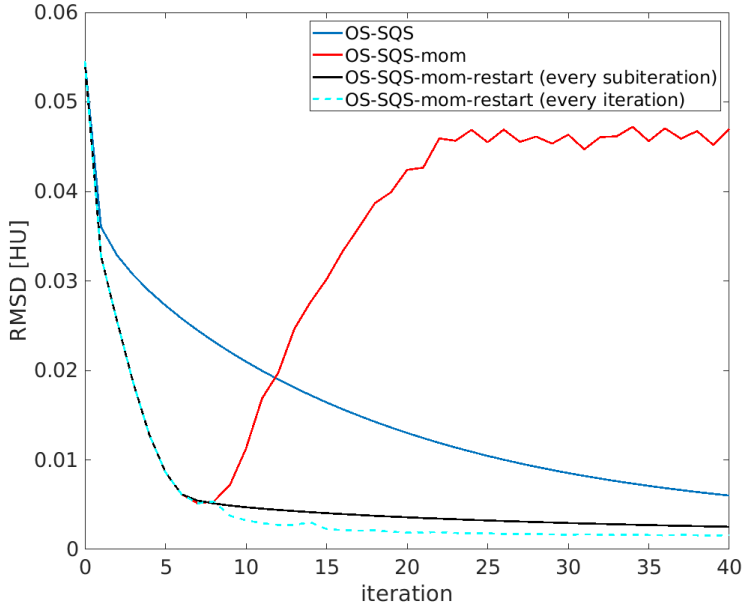
$\quad \boldsymbol{z}^{(\frac{k+1}{M})} = \boldsymbol{x}^{(\frac{k+1}{M})} + \frac{t_k - 1}{t_{k+1}}\left(\boldsymbol{x}^{(\frac{k+1}{M})} - \boldsymbol{x}^{(\frac{k}{M})}\right)$

Experimental setup - CT fan beam reconstruction

- Image size 128×128
- Sinogram size 222×50
- Hyperbola function used as the regularizer potential function
- Reference image was generated by running 5000 iterations of the 1 subset version of OS-SQS algorithm.
- regularization parameter = 16
- At iteration k, root mean square difference (RMSD) $= \sqrt{\frac{1}{|\Omega|} \sum_{j \in \Omega} |x_j^{(k)} - \hat{x}_j|^2}$

  $\Omega =$ cylindrical region-of-interest

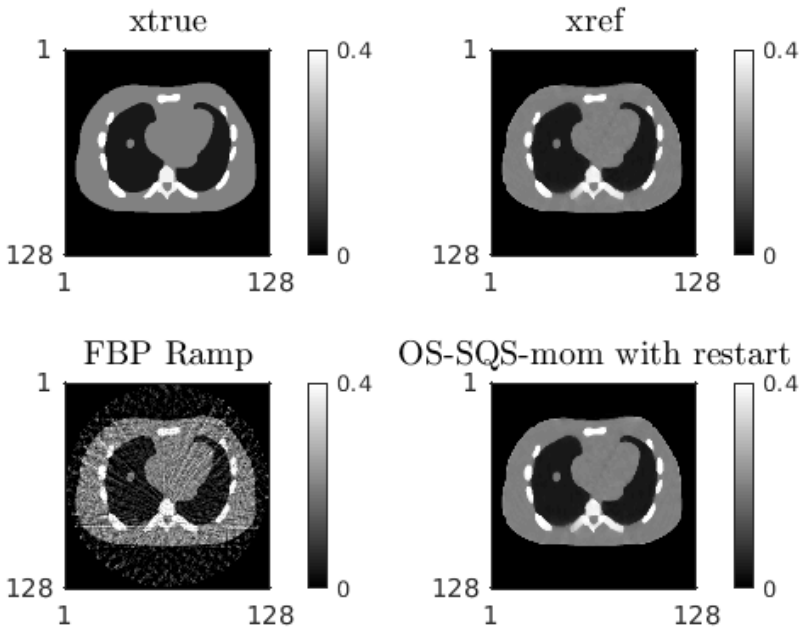
RMSD plots



- All algorithms were run with 8 subsets. Restart scheme was function-based.
- Note: Using momentum causes plot to diverge; restart helps stabilize it.

Errata: RMSD is in units of inverse length (1/cm), *not* in HU.

Reconstructed images



- $\boldsymbol{x}_{\text{ref}}$ = reference image; initial image $\boldsymbol{x}_0$ = FBP ramp image

8.6 Summary

- For large-scale machine learning problems, the SG method (or one of its variants) is a popular optimization approach.
- SG methods are applicable to a wide variety of smooth/non-smooth, and convex/non-convex problems.
- Some broad approaches to improve SG convergence:
  - Reduce variance of the stochastic estimate (keeping the learning rate fixed).
  - Design better learning rate schedules, using momentum methods or adaptive methods like Adagrad, RMSprop and Adam.
- Caveat: Algorithm hyperparameter tuning can be difficult for many of these algorithms (in contrast with GD, POGM, NCG etc.)

This is an active research area; *e.g.*, see [11] and these additional topics:

- Proximal subgradient methods
- Primal-dual subgradient methods (dual averaging) [51]
- Computer-aided analysis [52]
- Universal catalysts [53]
- Optimal methods [54] [55]
- Online methods [56] [57]

Caution: "(**SG**) method" might mean a **subgradient method**, or a **stochastic gradient method** (or a combination of both).

Bibliography

[1] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization methods for large-scale machine learning". In: *SIAM Review* 60.2 (2018), 223–311 (cit. on pp. 8.2, 8.19, 8.26).

[2] K. Slavakis, S-J. Kim, G. Mateos, and G. B. Giannakis. "Stochastic approximation vis-a-vis online learning for big data analytics". In: *IEEE Sig. Proc. Mag.* 31.6 (Nov. 2014), 124–9 (cit. on pp. 8.2, 8.21).

[3] J. Deng, W. Dong, R. Socher, L-J. Li, K. Li, and F-F. Li. "ImageNet: A large-scale hierarchical image database". In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2009, 248–55 (cit. on p. 8.4).

[4] B. Grimmer. "Convergence rates for deterministic and stochastic subgradient methods without Lipschitz continuity". In: *SIAM J. Optim.* 29.2 (2019), 1350–65 (cit. on pp. 8.5, 8.10, 8.13).

[5] D. H. Gutman and J. F. Pena. "Convergence rates of proximal gradient methods via the convex conjugate". In: *SIAM J. Optim.* 29.1 (2019), 162–74 (cit. on pp. 8.5, 8.10, 8.14, 8.16).

[6] N. Z. Shor. *Minimization methods for non-differentiable functions*. Springer-Verlag, 1985 (cit. on pp. 8.5, 8.6, 8.7, 8.8, 8.10, 8.11, 8.12).

[7] W. Yin. *Coordinate update algorithm short course: Subgradients and subgradient methods*. 2016 (cit. on pp. 8.7, 8.8).

[8] D. Ward. "Chain rules for nonsmooth functions". In: *J. Math. Anal. Applic.* 158.2 (1991), 519–38 (cit. on p. 8.8).

[9] A. Nedic and D. P. Bertsekas. "Incremental subgradient methods for nondifferentiable optimization". In: *SIAM J. Optim.* 12.1 (2001), 109–38 (cit. on p. 8.21).

[10] Y. Nesterov. "Smooth minimization of non-smooth functions". In: *Mathematical Programming* 103.1 (May 2005), 127–52 (cit. on p. 8.16).

[11] M. Ahookhosh and A. Neumaier. "Optimal subgradient algorithms for large-scale convex optimization in simple domains". In: *Numer. Algorithms* 76.4 (Dec. 2017), 1071–97 (cit. on pp. 8.16, 8.50).

[12] Y. Xu, Q. Lin, and T. Yang. *Accelerate stochastic subgradient method by leveraging local error bound*. 2016 (cit. on p. 8.16).

[13] Y. Xu, Y. Yan, Q. Lin, and T. Yang. *Homotopy smoothing for non-smooth problems with lower complexity than* $1/\epsilon$. 2016 (cit. on p. 8.16).

[14] T. Yang, Q. Lin, and Z. Li. *Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization*. 2016 (cit. on p. 8.16).

[15] Y. Yan, T. Yang, Z. Li, Q. Lin, and Y. Yang. "A unified analysis of stochastic momentum methods for deep learning". In: *ijcai*. 2018, 2955–61 (cit. on p. 8.16).

[16] T. Yang and Q. Lin. "RSG: beating subgradient method without smoothness and strong convexity". In: *J. Mach. Learning Res.* 19.6 (2018), 1–33 (cit. on p. 8.16).

[17] D. P. Bertsekas. *Incremental gradient, subgradient, and proximal methods for convex optimization: A survey*. August 2010 (revised December 2010) Report LIDS 2848. 2010 (cit. on pp. 8.16, 8.21).

[18] I. Konnov. "A non-monotone conjugate subgradient type method for minimization of convex functions". In: *J. Optim. Theory Appl.* 184.2 (Feb. 2020), 534–46 (cit. on p. 8.16).

[19] Y. Nesterov and V. Shikhman. "Quasi-monotone subgradient methods for nonsmooth convex minimization". In: *J. Optim. Theory Appl.* 165.3 (June 2015), 917–40 (cit. on p. 8.16).

[20] V. M. Kibardin. "Decomposition into functions in the minimization problem". In: *Avtomatika i Telemekhanika* 9 (Sept. 1979). Translation: p. 1311-23 in Plenum Publishing Co. "Adaptive Systems", 66–79 (cit. on p. 8.21).

[21] D. P. Bertsekas. "A new class of incremental gradient methods for least squares problems". In: *SIAM J. Optim.* 7.4 (Nov. 1997), 913–26 (cit. on p. 8.21).

[22] P. Tseng. "An incremental gradient(-projection) method with momentum term and adaptive stepsize rule". In: *SIAM J. Optim.* 8.2 (1998), 506–31 (cit. on p. 8.21).

[23] A. Geary and D. P. Bertsekas. "Incremental subgradient methods for nondifferentiable optimization". In: *Proc. Conf. Decision and Control*. Vol. 1. 1999, 907–12 (cit. on p. 8.21).

[24] K. C. Kiwiel. "Convergence of approximate and incremental subgradient methods for convex optimization". In: *SIAM J. Optim.* 14.3 (2004), 807–40 (cit. on p. 8.21).

[25] E. Salomão Helou-Neto and Álvaro Rodolfo De Pierro. "Incremental subgradients for constrained convex optimization: A unified framework and new methods". In: *SIAM J. Optim.* 20.3 (2009), 1547–72 (cit. on p. 8.21).

[26] D. P. Bertsekas. "Incremental proximal methods for large scale convex optimization". In: *Mathematical Programming* 129.2 (Oct. 2011), 163–95 (cit. on p. 8.21).

[27] S. Ahn, J. A. Fessler, D. Blatt, and A. O. Hero. "Convergent incremental optimization transfer algorithms: Application to tomography". In: *IEEE Trans. Med. Imag.* 25.3 (Mar. 2006), 283–96 (cit. on p. 8.21).

[28] Z. Q. Luo. "On the convergence of the LMS algorithm with adaptive learning rate for linear feedforward networks". In: *Neural Computation* 32.2 (June 1991), 226–45 (cit. on p. 8.21).

[29] D. Blatt, A. O. Hero, and H. Gauchman. "A convergent incremental gradient method with a constant step size". In: *SIAM J. Optim.* 18.1 (2007), 29–51.

[30] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo. "On the convergence rate of incremental aggregated gradient algorithms". In: *SIAM J. Optim.* 27.2 (Jan. 2017), 1035–48.

[31] N. D. Vanli, M. Gurbuzbalaban, and A. Ozdaglar. "Global convergence rate of incremental aggregated gradient methods for nonsmooth problems". In: *Proc. Conf. Decision and Control*. 2016, 173–8.

[32] N. D. Vanli, M. Gurbuzbalaban, and A. Ozdaglar. "Global convergence rate of proximal incremental aggregated gradient methods". In: *SIAM J. Optim.* 28.2 (Jan. 2018), 1282–300.

[33] X. Zhang, W. Peng, H. Zhang, and W. Zhu. *Inertial proximal incremental aggregated gradient method*. 2017.

[34] A. Mokhtari, M. Gurbuzbalaban, and A. Ribeiro. "A double incremental aggregated gradient method with linear convergence rate for large-scale optimization". In: *Proc. IEEE Conf. Acoust. Speech Sig. Proc.* 2017, 4696–700.

[35] A. Mokhtari, M. Gurbuzbalaban, and A. Ribeiro. "Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate". In: *SIAM J. Optim.* 28.2 (Jan. 2018), 1420–47.

[36] A. Defazio, F. Bach, and S. Lacoste-Julien. "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives". In: *Neural Info. Proc. Sys.* 2014, 1646–54 (cit. on p. 8.30).

[37] R. Johnson and T. Zhang. "Accelerating stochastic gradient descent using predictive variance reduction". In: *Neural Info. Proc. Sys.* 2013, 315–23 (cit. on p. 8.30).

[38] B. Li and G. B. Giannakis. *Adaptive step sizes in variance reduction via regularization*. 2019 (cit. on p. 8.30).

[39] S. Ruder. *An overview of gradient descent optimization algorithms*. 2017 (cit. on pp. 8.30, 8.35).

[40] I. Loshchilov and F. Hutter. "SGDR: stochastic gradient descent with warm restarts". In: *Proc. Intl. Conf. on Learning Representations*. 2017 (cit. on p. 8.30).

[41] M. Schmidt, N. Le Roux, and F. Bach. "Minimizing finite sums with the stochastic average gradient". In: *Mathematical Programming* 162.1 (Mar. 2017), 83–112 (cit. on p. 8.34).

[42] B. O'Donoghue and E. Candes. "Adaptive restart for accelerated gradient schemes". In: *Found. Comp. Math.* 15.3 (June 2015), 715–32 (cit. on p. 8.39).

[43] N. Azizan, S. Lale, and B. Hassibi. *Stochastic mirror descent on overparameterized nonlinear models: convergence, implicit regularization, and generalization*. ICML. 2019.

[44] P. L. Bartlett, P. M. Long, Gabor Lugosi, and A. Tsigler. *Benign overfitting in linear regression*. 2019.

[45] M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine learning practice and the bias-variance trade-off". In: *Proc. Natl. Acad. Sci.* 116.32 (Aug. 2019), 15849–54.

[46] S. S. Du, X. Zhai, B. Poczos, and A. Singh. "Gradient descent provably optimizes over-parameterized neural networks". In: *Proc. Intl. Conf. on Learning Representations*. 2019.

[47] S. Mei and A. Montanari. *The generalization error of random features regression: Precise asymptotics and double descent curve*. 2019.

[48] N. Azizan and B. Hassibi. *Stochastic gradient/mirror descent: minimax optimality and implicit regularization*. 2018.

[49] D. P. Kingma and J. Ba. *Adam: A method for stochastic optimization*. 2014 (cit. on p. 8.40).

[50] D. Kim, S. Ramani, and J. A. Fessler. "Combining ordered subsets and momentum for accelerated X-ray CT image reconstruction". In: *IEEE Trans. Med. Imag.* 34.1 (Jan. 2015), 167–78 (cit. on pp. 8.41, 8.45).

[51] Y. Nesterov. "Primal-dual subgradient methods for convex problems". In: *Mathematical Programming* 120.1 (2009), 261–83 (cit. on p. 8.50).

[52] A. Taylor and F. Bach. "Stochastic first-order methods: non-asymptotic and computer-aided analyses via potential functions". In: *colt*. 2019, 2934–92 (cit. on p. 8.50).

[53] H. Lin, J. Mairal, and Z. Harchaoui. "A universal catalyst for first-order optimization". In: *Neural Info. Proc. Sys.* 2015, 3384–92 (cit. on p. 8.50).

[54] N. S. Aybat, A. Fallah, M. Gurbuzbalaban, and A. Ozdaglar. *A universally optimal multistage accelerated stochastic gradient method*. 2019 (cit. on p. 8.50).

[55] A. Jofre and P. Thompson. "On variance reduction for stochastic smooth convex optimization with multiplicative noise". In: *Mathematical Programming* 174.1-2 (Mar. 2019), 253–92 (cit. on p. 8.50).

[56] L. Xiao. "Dual averaging methods for regularized stochastic learning and online optimization". In: *J. Mach. Learning Res.* 11 (Dec. 2010), 2543–96 (cit. on p. 8.50).

[57] J. C. Duchi, A. Agarwal, and M. J. Wainwright. "Dual averaging for distributed optimization: convergence analysis and network scaling". In: *IEEE Trans. Auto. Control* 57.3 (Mar. 2012), 592–606 (cit. on p. 8.50).