Large-Scale Optimization for Machine Learning

Julien Mairal
Inria Grenoble
Optimization is central to machine learning

In supervised learning, we learn a **prediction function** $h : \mathcal{X} \to \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1,\ldots,n}$ with $x_i$ in $\mathcal{X}$, and $y_i$ in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).$$

- **Empirical risk, data fit**
- **Regularization**

Julien Mairal

Large-scale optimization for machine learning
Optimization is central to machine learning

In supervised learning, we learn a prediction function \( h : \mathcal{X} \to \mathcal{Y} \) given labeled training data \((x_i, y_i)_{i=1,...,n}\) with \(x_i\) in \(\mathcal{X}\), and \(y_i\) in \(\mathcal{Y}\):

\[
\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).
\]

The labels \(y_i\) are in

- \([-1, +1]\) for **binary** classification.
- \(\{1, \ldots, K\}\) for **multi-class** classification.
- \(\mathbb{R}\) for **regression**.
- \(\mathbb{R}^k\) for **multivariate regression**.
- any general set for **structured prediction**.
Optimization is central to machine learning

In supervised learning, we learn a prediction function $h : \mathcal{X} \to \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1,\ldots,n}$ with $x_i$ in $\mathcal{X}$, and $y_i$ in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).$$

**empirical risk, data fit**
**regularization**

Example with linear models: logistic regression, SVMs, etc.

- assume there exists a linear relation between $y$ and features $x$ in $\mathbb{R}^p$.
- $h(x) = w^\top x + b$ is parametrized by $w, b$ in $\mathbb{R}^{p+1}$.
- $L$ is often a convex loss function.
- $\Omega(h)$ is often the squared $\ell_2$-norm $\|w\|^2$. 
Optimization is central to machine learning

A few examples of linear models with no bias $b$:

**Ridge regression:**
$$
\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2.
$$

**Linear SVM:**
$$
\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i w^T x_i) + \lambda \|w\|_2^2.
$$

**Logistic regression:**
$$
\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \log \left( 1 + e^{-y_i w^T x_i} \right) + \lambda \|w\|_2^2.
$$
Optimization is central to machine learning

In supervised learning, we learn a prediction function \( h : \mathbb{R}^p \rightarrow \mathcal{Y} \) given labeled training data \((x_i, y_i)_{i=1,...,n}\) with \( x_i \) in \( \mathbb{R}^p \), and \( y_i \) in \( \mathcal{Y} \):

\[
\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).
\]

empirical risk, data fit

regularization

What is specific to multilayer neural networks?

- The “neural network” space \( \mathcal{H} \) is explicitly parametrized by:

\[
h(x) = \sigma_k(A_k \sigma_{k-1}(A_{k-1} \ldots \sigma_2(A_2 \sigma_1(A_1x)) \ldots)).
\]

- Linear operations are either unconstrained or they share parameters (e.g., convolutions).

- Finding the optimal \( A_1, A_2, \ldots, A_k \) yields a non-convex problem in huge dimension.
Optimization is central to machine learning

In supervised learning, we learn a prediction function $h : \mathbb{R}^p \to \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1,...,n}$ with $x_i$ in $\mathbb{R}^p$, and $y_i$ in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).$$

Even with simple linear models, it leads to challenging problems in optimization:

- **scaling** both in the problem size $n$ and dimension $p$;
- being able to **exploit the problem structure** (finite sum);
- obtaining **convergence and numerical stability** guarantees;
- obtaining **statistical guarantees**.
Optimization is central to machine learning

In supervised learning, we learn a **prediction function** $h : \mathbb{R}^p \to \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1,...,n}$ with $x_i$ in $\mathbb{R}^p$, and $y_i$ in $\mathcal{Y}$:

$$
\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)) + \lambda \Omega(h).
$$

It is not limited to supervised learning

$$
\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i)) + \lambda \Omega(h).
$$

- $L$ is not a classification loss any more;
- K-means, PCA, EM with mixture of Gaussian, matrix factorization, auto-encoders... can be explained with such a formulation.
What would be a great outline for this course

1. **Statistical learning** and empirical risk minimization.
2. General principles of *gradient-based* optimization.
   - convex optimization
   - non-convex optimization
   - non-smooth optimization
3. **Quasi-Newton** methods.
4. **Stochastic** Optimization.
5. **Distributed** Optimization.
6. ...
What we will do

- Introduction to statistical learning and gradient-based optimization.
- Introduction to **stochastic** optimization.
- Two advanced topics:
  - Variance-reduced stochastic gradient descent.
  - Nesterov’s acceleration (momentum).
Part I: Statistical learning and gradient-based optimization
Statistical learning

Setting

- We draw i.i.d. pairs $(x_i, y_i)$ from some unknown distribution $P$.
- The objective is to minimize over all functions the expected risk:

$$\min_h \left\{ R(h) = \mathbb{E}_{(x,y) \sim P} [L(y, h(x))] \right\}.$$
Statistical learning

Setting

- We draw i.i.d. pairs \((x_i, y_i)\) from some unknown distribution \(P\).
- The objective is to minimize over all functions the expected risk:

\[
\min_h \left\{ R(h) = \mathbb{E}_{(x,y) \sim P} [L(y, h(x))] \right\}.
\]

But
Statistical learning

Setting

- We draw i.i.d. pairs \((x_i, y_i)\) from some unknown distribution \(P\).
- The objective is to minimize over all functions the expected risk:

  \[
  \min_H \left\{ R(h) = \mathbb{E}_{(x,y) \sim P}[L(y, h(x))] \right\}.
  \]

But

- we do minimize over a class of functions \(\mathcal{H}\) only.
Statistical learning

Setting

- We draw i.i.d. pairs \((x_i, y_i)\) from some unknown distribution \(P\).
- The objective is to minimize over all functions the expected risk:

\[
\min_h \left\{ R(h) = \mathbb{E}_{(x,y) \sim P}[L(y, h(x))] \right\}.
\]

But

1. We do minimize over a class of functions \(\mathcal{H}\) only.
2. Datasets are often finite and we minimize instead the empirical risk:

\[
\min_{h \in \mathcal{H}} \left\{ R_n(h) = \frac{1}{n} \sum_{i=1}^{n} [L(y_i, h(x_i))] \right\}.
\]
Statistical learning

Setting

- We draw i.i.d. pairs $(x_i, y_i)$ from some unknown distribution $P$.
- The objective is to minimize over all functions the expected risk:
  \[
  \min_h \left\{ R(h) = \mathbb{E}_{(x,y) \sim P}[L(y, h(x))] \right\}.
  \]

But

1. we do minimize over a class of functions $\mathcal{H}$ only.
2. datasets are often finite and we minimize instead the empirical risk:
  \[
  \min_{h \in \mathcal{H}} \left\{ R_n(h) = \frac{1}{n} \sum_{i=1}^{n} [L(y_i, h(x_i))] \right\}.
  \]
3. we minimize approximately.
Statistical learning

\[ \hat{h}_n \in \arg \min_{h \in \mathcal{H}} R_n(h). \]

**Approximation/Estimation:**

\[
R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h) = R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h) + \min_{h \in \mathcal{H}} R(h) - \min_{h} R(h)
\]

- estimation error
- approximation error

- Controlled with **regularization** (bias/variance, over/under-fitting)
Statistical learning

$$\hat{h}_n \in \arg \min_{h \in \mathcal{H}} R_n(h).$$

**Approximation/Estimation/Optimization:**

$$R(\hat{h}_n) - \min_h R(h) = R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h) + \min_{h \in \mathcal{H}} R(h) - \min_h R(h)$$

- **estimation error**
- **approximation error**

- Controlled with **regularization** (bias/variance, over/under-fitting)
- $\hat{h}_n$ is obtained **approximately** by optimization:

$$R(\tilde{h}_n) - \min_h R(h) = R(\tilde{h}_n) - R(\hat{h}_n) + R(\hat{h}_n) - \min_h R(h)$$

- **optimization error**

- Insight of Bottou and Bousquet (2008): **no need to optimize below statistical error!**
Statistical learning

Classical rates of estimation

- $O(D(\mathcal{H})/\sqrt{n})$ with $D(\mathcal{H})$ growing with the class of function $\mathcal{H}$.
- under specific conditions, faster rates may be achieved $O(1/n)$.

Statistical learning

Classical rates of estimation

- $O(D(\mathcal{H})/\sqrt{n})$ with $D(\mathcal{H})$ growing with the class of function $\mathcal{H}$.
- under specific conditions, faster rates may be achieved $O(1/n)$.


What conclusions can we draw from an optimization perspective?

- convergence rate of stochastic gradient descent (at least for convex problems) may be **asymptotically optimal**.
Statistical learning

Classical rates of estimation

- $O(D(H)/\sqrt{n})$ with $D(H)$ growing with the class of function $H$.
- under specific conditions, faster rates may be achieved $O(1/n)$.


What conclusions can we draw from an optimization perspective?

- convergence rate of stochastic gradient descent (at least for convex problems) may be asymptotically optimal.
- faster algorithms than SGD are not always useful, but they are if
  - they are easier to use than SGD (no parameter tuning).
  - if forgetting the initial condition with SGD takes time (hard to know in advance).
**Statistical learning**

**Classical rates of estimation**
- \( O(D(\mathcal{H})/\sqrt{n}) \) with \( D(\mathcal{H}) \) growing with the class of function \( \mathcal{H} \).
- under specific conditions, faster rates may be achieved \( O(1/n) \).


**What conclusions can we draw from an optimization perspective?**
- convergence rate of stochastic gradient descent (at least for convex problems) may be **asymptotically optimal**.
- faster algorithms than SGD are not always useful, but they are if
  - they are easier to use than SGD (**no parameter tuning**).
  - if forgetting the initial condition with SGD takes time (**hard to know in advance**).
- **mathematics, engineering, and experiments are needed.**
An important quantity to quantify smoothness is the **Lipschitz constant** of the gradient:

\[ \| \nabla f(x) - \nabla f(y) \| \leq L \| x - y \|. \]
Basics of gradient-based optimization

Smooth vs non-smooth

An important quantity to quantify smoothness is the **Lipschitz constant** of the gradient:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$  

If $f$ is twice differentiable, $L$ may be chosen as the **largest eigenvalue** of the Hessian $\nabla^2 f$. This is an upper-bound on the function curvature.
An important quantity to quantify convexity is the **strong-convexity** constant

\[
f(x) \geq f(y) + \nabla f(y)^\top (x - y) + \frac{\mu}{2} \|x - y\|^2,
\]

where \(\mu\) is the strong-convexity constant.
Basic of gradient-based optimization

Convex vs non-convex

An important quantity to quantify convexity is the strong-convexity constant

\[ f(x) \geq f(y) + \nabla f(y)^\top (x - y) + \frac{\mu}{2} \| x - y \|^2, \]

If \( f \) is twice differentiable, \( \mu \) may be chosen as the smallest eigenvalue of the Hessian \( \nabla^2 f \). This is a lower-bound on the function curvature.
Basics of gradient-based optimization
Convex Functions

Why do we care about convexity?
Basics of gradient-based optimization

Convex Functions

Local observations give information about the global optimum

- $\nabla f(x) = 0$ is a necessary and sufficient optimality condition for differentiable convex functions;
- it is often easy to upper-bound $f(x) - f^*$. 
Basics of gradient-based optimization

If $f$ is convex and smooth

\[ f(x) \geq f(x_0) + \nabla f(x_0)^\top (x - x_0); \]

linear approximation

- if $f$ is non-smooth, a similar inequality holds for subgradients.
Basics of gradient-based optimization

If $\nabla f$ is $L$-Lipschitz continuous ($f$ does not need to be convex)

1. $f(x) \leq g(x) = f(x_0) + \nabla f(x_0)^\top (x - x_0) + \frac{L}{2} \|x - x_0\|_2^2$; linear approximation
Basics of gradient-based optimization

If $\nabla f$ is $L$-Lipschitz continuous ($f$ does not need to be convex)

\[ f(x) \leq g(x) = f(x_0) + \nabla f(x_0) \top (x - x_0) + \frac{L}{2} \|x - x_0\|^2_2; \]

\[ g(x) = Cx_0 + \frac{L}{2} \|x_0 - (1/L)\nabla f(x_0) - x\|^2_2. \]
Basics of gradient-based optimization

If $\nabla f$ is $L$-Lipschitz continuous ($f$ does not need to be convex)

- $f(x) \leq g(x) = f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{L}{2} ||x - x_0||_2^2$; linear approximation
- $x_1 = x_0 - \frac{1}{L} \nabla f(x_0)$ (gradient descent step).
Assume that $f$ is convex and $L$-smooth ($\nabla f$ is $L$-Lipschitz).

**Theorem**

Consider the algorithm

$$x_t \leftarrow x_{t-1} - \frac{1}{L} \nabla f(x_{t-1}).$$

Then,

$$f(x_t) - f^* \leq \frac{L \|x_0 - x^*\|_2^2}{2t}.$$
Basics of gradient-based optimization

Gradient descent algorithm

Assume that $f$ is convex and $L$-smooth ($\nabla f$ is $L$-Lipschitz).

**Theorem**

Consider the algorithm

$$x_t \leftarrow x_{t-1} - \frac{1}{L} \nabla f(x_{t-1}).$$

Then,

$$f(x_t) - f^* \leq \frac{L\|x_0 - x^*\|_2^2}{2t}.$$ 

How to prove this?

**Read Nesterov's book!** [Nesterov, 2004].
Proof (1/2)

Proof of the main inequality for smooth functions

We want to show that for all $x$ and $z$,

$$f(x) = f(z) + \nabla f(z) \top (x - z) + \frac{L}{2} \|x - z\|_2^2.$$
Proof (1/2)

Proof of the main inequality for smooth functions

We want to show that for all $x$ and $z$,

$$f(x) \leq f(z) + \nabla f(z)^\top (x - z) + \frac{L}{2} \|x - z\|^2.$$

By using Taylor’s theorem with integral form,

$$f(x) - f(z) = \int_0^1 \nabla f(tx + (1 - t)z)^\top (x - z) dt.$$
Proof (1/2)

Proof of the main inequality for smooth functions

We want to show that for all $x$ and $z$,

$$f(x) \leq f(z) + \nabla f(z)^\top (x - z) + \frac{L}{2} \|x - z\|^2_2.$$

By using Taylor’s theorem with integral form,

$$f(x) - f(z) = \int_0^1 \nabla f(tx + (1 - t)z)^\top (x - z)\,dt.$$

Then,

$$f(x) - f(z) - \nabla f(z)^\top (x - z) = \int_0^1 (\nabla f(tx + (1 - t)z) - \nabla f(z))^\top (x - z)\,dt$$

$$\leq \int_0^1 |(\nabla f(tx + (1 - t)z) - \nabla f(z))^\top (x - z)|\,dt$$

$$\leq \int_0^1 \|\nabla f(tx + (1 - t)z) - \nabla f(z)\|_2 \|x - z\|_2\,dt \quad \text{(C.-S.)}$$

$$\leq \int_0^1 Lt\|x - z\|_2^2\,dt = \frac{L}{2} \|x - z\|_2^2.$$
Proof (2/2)

Proof of the theorem

We have shown that for all $x$,

\[
f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^T (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|^2.
\]

$g_t$ is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|^2$. Then,
Proof (2/2)

Proof of the theorem

We have shown that for all $x$,

$$f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^\top (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|^2_2.$$

$g_t$ is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|^2_2$. Then,

$$f(x_t) \leq g_t(x_t)$$
Proof (2/2)

Proof of the theorem

We have shown that for all $x$,

$$f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^T (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|^2_2.$$  

$g_t$ is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|^2_2$. Then,

$$f(x_t) \leq g_t(x_t) = g_t(x^*) - \frac{L}{2} \|x^* - x_t\|^2_2.$$
Proof (2/2)

Proof of the theorem

We have shown that for all $x$,

$$f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^\top (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|^2_2.$$ 

$g_t$ is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|^2_2$. Then,

$$f(x_t) \leq g_t(x_t) = g_t(x^*) - \frac{L}{2} \|x^* - x_t\|^2_2$$

$$= f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|^2_2 - \frac{L}{2} \|x^* - x_t\|^2_2.$$
Proof (2/2)
Proof of the theorem

We have shown that for all $x$,

$$f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^\top (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|_2^2.$$

$g_t$ is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|_2^2$. Then,

$$f(x_t) \leq g_t(x_t) = g_t(x^*) - \frac{L}{2} \|x^* - x_t\|_2^2$$

$$= f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2$$

$$\leq f^* + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2.$$
Proof (2/2)

Proof of the theorem

We have shown that for all \( x \),

\[
f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^\top (x - x_{t-1}) + \frac{L}{2} \|x - x_{t-1}\|_2^2.
\]

\( g_t \) is minimized by \( x_t \); it can be rewritten \( g_t(x) = g_t(x_t) + \frac{L}{2} \|x - x_t\|_2^2 \). Then,

\[
f(x_t) \leq g_t(x_t) = g_t(x^*) - \frac{L}{2} \|x^* - x_t\|_2^2
\]

\[
= f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2
\]

\[
\leq f^* + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2.
\]

By summing from \( t = 1 \) to \( T \), we have a telescopic sum

\[
T(f(x_T) - f^*) \leq \sum_{t=1}^{T} f(x_t) - f^* \leq \frac{L}{2} \|x^* - x_0\|_2^2 - \frac{L}{2} \|x^* - x_T\|_2^2 \leq \frac{L}{2} \|x^* - x_0\|_2^2.
\]
Proof (2/2)

Proof of the theorem

We have shown that for all $x$,

$$ f(x) \leq g_t(x) = f(x_{t-1}) + \nabla f(x_{t-1})^T (x - x_{t-1}) + \frac{L}{2} \| x - x_{t-1} \|_2^2. $$

g_t is minimized by $x_t$; it can be rewritten $g_t(x) = g_t(x_t) + \frac{L}{2} \| x - x_t \|_2^2$. Then,

$$ f(x_t) \leq g_t(x_t) = g_t(x^*) - \frac{L}{2} \| x^* - x_t \|_2^2 $$

$$ = f(x_{t-1}) + \nabla f(x_{t-1})^T (x^* - x_{t-1}) + \frac{L}{2} \| x^* - x_{t-1} \|_2^2 - \frac{L}{2} \| x^* - x_t \|_2^2 $$

$$ \leq f^* + \frac{L}{2} \| x^* - x_{t-1} \|_2^2 - \frac{L}{2} \| x^* - x_t \|_2^2. $$

By summing from $t = 1$ to $T$, we have a telescopic sum

$$ T(f(x_T) - f^*) \leq \sum_{t=1}^{T} f(x_t) - f^* \leq \frac{L}{2} \| x^* - x_0 \|_2^2 - \frac{L}{2} \| x^* - x_T \|_2^2 \leq \frac{L}{2} \| x^* - x_0 \|_2^2. $$

(green) - (red) - (blue) - telescopic sum
Basics of gradient-based optimization

If $\nabla f$ is $L$-Lipschitz continuous and $f$ $\mu$-strongly convex

\[ f(x) \leq f(x_0) + \nabla f(x_0)^\top (x - x_0) + \frac{L}{2} \|x - x_0\|_2^2; \]
\[ f(x) \geq f(x_0) + \nabla f(x_0)^\top (x - x_0) + \frac{\mu}{2} \|x - x_0\|_2^2; \]
Basics of gradient-based optimization

Proposition

When $f$ is $\mu$-strongly convex and $L$-smooth, the gradient descent algorithm with step-size $1/L$ produces iterates such that

$$f(x_t) - f^* \leq \left(1 - \frac{\mu}{L}\right)^t \frac{L\|x_0 - x^*\|_2^2}{2}.$$ 

We call that a linear convergence rate.

Remarks

- if $f$ is twice differentiable, $L$ and $\mu$ represent the largest and smallest eigenvalues of the Hessian, respectively.
- $L/\mu$ is called the condition number.
Basics of gradient-based optimization

Picture from F. Bach

(large $\mu/L$)  (small $\mu/L$)
Proof

We start from a (blue) inequality from the previous proof

\[
    f(x_t) \leq f(x_{t-1}) + \nabla f(x_{t-1})^T (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2
\]

\[
    \leq f^* + \frac{L - \mu}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2.
\]
Proof

We start from a (blue) inequality from the previous proof

\[
    f(x_t) \leq f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2
\]

\[
    \leq f^* + \frac{L - \mu}{2} \|x^* - x_{t-1}\|_2^2 - \frac{L}{2} \|x^* - x_t\|_2^2.
\]

In addition, blue! \( f(x_t) \geq f^* + \frac{\mu}{2} \|x_t - x^*\|_2^2 \), and thus

\[
    \|x^* - x_t\|_2^2 \leq \frac{L - \mu}{L + \mu} \|x^* - x_{t-1}\|_2^2
\]

\[
    \leq \left(1 - \frac{\mu}{L}\right) \|x^* - x_{t-1}\|_2^2.
\]
Proof

We start from a (blue) inequality from the previous proof

\[
f(x_t) \leq f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|^2_2 - \frac{L}{2} \|x^* - x_t\|^2_2
\]

\[
\leq f^* + \frac{L - \mu}{2} \|x^* - x_{t-1}\|^2_2 - \frac{L}{2} \|x^* - x_t\|^2_2.
\]

In addition, blue! \(f(x_t) \geq f^* + \frac{\mu}{2} \|x_t - x^*\|^2_2\), and thus

\[
\|x^* - x_t\|^2_2 \leq \frac{L - \mu}{L + \mu} \|x^* - x_{t-1}\|^2_2 \\
\leq \left(1 - \frac{\mu}{L}\right) \|x^* - x_{t-1}\|^2_2.
\]

Finally, green! \(f(x_t) \leq f^* + \nabla f(x^*)^\top (x_t - x^*) + \frac{L}{2} \|x_t - x^*\|^2\) with \(\nabla f(x^*) = 0\):

\[
f(x_t) - f^* \leq \frac{L}{2} \|x_t - x^*\|^2_2 \leq \left(1 - \frac{\mu}{L}\right)^t \frac{L \|x^* - x_0\|^2_2}{2}
\]
Proof

We start from a (blue) inequality from the previous proof

\[ f(x_t) \leq f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^* - x_{t-1}) + \frac{L}{2} \|x^* - x_{t-1}\|^2_2 - \frac{L}{2} \|x^* - x_t\|^2_2 \]

\[ \leq f^* + \frac{L - \mu}{2} \|x^* - x_{t-1}\|^2_2 - \frac{L}{2} \|x^* - x_t\|^2_2. \]

In addition, blue! \( f(x_t) \geq f^* + \frac{\mu}{2} \|x_t - x^*\|^2_2 \), and thus

\[ \|x^* - x_t\|^2_2 \leq \frac{L - \mu}{L + \mu} \|x^* - x_{t-1}\|^2_2 \]

\[ \leq \left( 1 - \frac{\mu}{L} \right) \|x^* - x_{t-1}\|^2_2. \]

Finally, green! \( f(x_t) \leq f^* + \nabla f(x^*)^\top (x_t - x^*) + \frac{L}{2} \|x_t - x^*\|^2 \) with \( \nabla f(x^*) = 0 \):

\[ f(x_t) - f^* \leq \frac{L}{2} \|x_t - x^*\|^2_2 \leq \left( 1 - \frac{\mu}{L} \right)^t \frac{L \|x^* - x_0\|^2_2}{2} \]

It is all about green and blue.
Basics of gradient-based optimization: composite problems

A **composite** optimization problem consists of minimizing the sum of a smooth and non-smooth function

\[
\min_{x \in \mathbb{R}^p} \{ f(x) = f_0(x) + \psi(x) \},
\]

where \( f_0 \) is \( L \)-smooth and \( \psi \) is convex but not necessarily smooth.

**Example**

A popular choice is \( \psi(x) = \|x\|_1 \), which induces sparsity.
Basics of gradient-based optimization: composite problems

A **composite** optimization problem consists of minimizing the sum of a smooth and non-smooth function

$$\min_{x \in \mathbb{R}^p} \{ f(x) = f_0(x) + \psi(x) \},$$

where $f_0$ is $L$-smooth and $\psi$ is convex but not necessarily smooth.

**Example**

A popular choice is $\psi(x) = \|x\|_1$, which induces sparsity.

Basics of gradient-based optimization: composite problems

Remark: with stepsize $1/L$, gradient descent may be interpreted as iteratively minimizing a tight upper-bound:

$$f(x) \leq g_t(x)$$

**Figure:** At each step, we update $x_t \in \arg\min_{x \in \mathbb{R}^p} g_t(x)$
Basics of gradient-based optimization: composite problems

An important inequality for composite functions

If $\nabla f_0$ is $L$-Lipschitz continuous

$$f(x) = f_0(x) + \psi(x)$$

- $f(x) \leq f_0(x_0) + \nabla f_0(x_0)^T (x - x_0) + \frac{L}{2} \|x - x_0\|^2 + \psi(x)$;
- $x_1$ minimizes $g$. 
Basics of gradient-based optimization: composite problems

Gradient descent for minimizing $f$ consists of

$$x_t \leftarrow \arg\min_{x \in \mathbb{R}^p} g_t(x) \iff x_t \leftarrow x_{t-1} - \frac{1}{L} \nabla f(x_{t-1}).$$

The proximal gradient method for minimizing $f = f_0 + \psi$ consists of

$$x_t \leftarrow \arg\min_{x \in \mathbb{R}^p} g_t(x),$$

which is equivalent to

$$x_t \leftarrow \arg\min_{x \in \mathbb{R}^p} \frac{1}{2} \left\| x_{t-1} - \frac{1}{L} \nabla f_0(x_{t-1}) - x \right\|^2_2 + \frac{1}{L} \psi(x).$$

It requires computing efficiently the **proximal operator** of $\psi$.

$$y \mapsto \arg\min_{x \in \mathbb{R}^p} \frac{1}{2} \left\| y - x \right\|^2_2 + \psi(x).$$
Basics of gradient-based optimization: composite problems

Remarks

- also known as forward-backward algorithm;
- same convergence rates as GD - same proofs;
- there exists line search schemes to automatically tune \( L \);
- proximal operator can be computed for many interesting functions.

The case of \( \ell_1 \)

The proximal operator of \( \lambda \| \cdot \|_1 \) is the soft-thresholding operator

\[
x[j] = \text{sign}(y[j])(|y[j]| - \lambda)^+.
\]

The resulting algorithm is called iterative soft-thresholding.

Accelerated gradient descent methods

Nesterov introduced in the 80’s an acceleration scheme for the gradient descent algorithm.

Generalization to the composite setting: FISTA

\[ x_t \leftarrow \arg \min_{x \in \mathbb{R}^p} \frac{1}{2} \left\| x - \left( y_{t-1} - \frac{1}{L} \nabla f_0(y_{t-1}) \right) \right\|_2^2 + \frac{1}{L} \psi(x); \]

Find \( \alpha_t > 0 \) s.t. \( \alpha_t^2 = (1 - \alpha_t) \alpha_{t-1}^2 + \frac{\mu}{L} \alpha_t; \)

\[ y_t \leftarrow x_t + \beta_t (x_t - x_{t-1}) \quad \text{with} \quad \beta_t = \frac{\alpha_{t-1} (1 - \alpha_{t-1})}{\alpha_{t-1}^2 + \alpha_t}. \]

- \( f(x_t) - f^* = O(1/t^2) \) for convex problems;
- \( f(x_t) - f^* = O((1 - \sqrt{\mu/L})^t) \) for \( \mu \)-strongly convex problems;
- Acceleration works in many practical cases.

What do we mean by “acceleration”?

Complexity analysis

The complexity to guarantee \( f(x_t) - f^* \leq \varepsilon \), is given below

<table>
<thead>
<tr>
<th></th>
<th>( \mu &gt; 0 )</th>
<th>( \mu = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISTA</td>
<td>( O \left( \frac{L}{\mu} \log \left( \frac{1}{\varepsilon} \right) \right) )</td>
<td>( O \left( \frac{L}{\varepsilon} \right) )</td>
</tr>
<tr>
<td>FISTA</td>
<td>( O \left( \sqrt{\frac{L}{\mu}} \log \left( \frac{1}{\varepsilon} \right) \right) )</td>
<td>( O \left( \sqrt{\frac{L}{\varepsilon}} \right) )</td>
</tr>
</tbody>
</table>

Remarks

- the rate of FISTA is optimal for a “first-order local black box” [Nesterov, 2004].
- for non-convex problems, acceleration often works in practice, but is poorly understood from a theoretical perspective (local convexity? convexity along trajectories? saddle-point escape?).
How does “acceleration” work?

Unfortunately, the literature does not provide any simple geometric explanation...

Simple gradient descent steps are “blind” to the past iterates, and are based on a purely local model of the objective. Accelerated methods usually involve an extrapolation step $y_t = x_t + \beta_t (x_t - x_{t-1})$ with $\beta_t$ in $(0,1)$. Nesterov interprets acceleration as relying on a better model of the objective called estimate sequence.
How does “acceleration” work?

Unfortunately, the literature does not provide any simple geometric explanation... but they are a few obvious facts and a mechanism introduced by Nesterov, called “estimate sequence”.

Obvious facts

- Simple gradient descent steps are “blind” to the past iterates, and are based on a purely local model of the objective.
- Accelerated methods usually involve an extrapolation step $y_t = x_t + \beta_t(x_t - x_{t-1})$ with $\beta_t$ in $(0, 1)$.
- Nesterov interprets acceleration as relying on a better model of the objective called estimate sequence.
How does “acceleration” work?

Definition of estimate sequence [Nesterov].

A pair of sequences \((\varphi_t)_{t \geq 0}\) and \((\lambda_t)_{t \geq 0}\), with \(\lambda_t \geq 0\) and \(\varphi_t : \mathbb{R}^p \to \mathbb{R}\), is called an estimate sequence of function \(f\) if \(\lambda_t \to 0\) and

\[
\text{for any } x \in \mathbb{R}^p \text{ and all } t \geq 0, \quad \varphi_t(x) - f(x) \leq \lambda_t(\varphi_0(x) - f(x)).
\]

In addition, if for some sequence \((x_t)_{t \geq 0}\) we have

\[
f(x_t) \leq \varphi_t^* \triangleq \min_{x \in \mathbb{R}^p} \varphi_t(x),
\]

then

\[
f(x_t) - f^* \leq \lambda_t(\varphi_0(x^*) - f^*),
\]

where \(x^*\) is a minimizer of \(f\).
How does “acceleration” work?

In summary, we need two properties

1. \( \varphi_t(x) \leq (1 - \lambda_t)f(x) + \lambda_t \varphi_0(x); \)
2. \( f(x_t) \leq \varphi^*_t \triangleq \min_{x \in \mathbb{R}^p} \varphi_t(x). \)

Remarks

- \( \varphi_t \) is neither an upper-bound, nor a lower-bound;
- Finding the right estimate sequence is often nontrivial.
How does “acceleration” work?

In summary, we need two properties

1. \( \phi_t(x) \leq (1 - \lambda_t)f(x) + \lambda_t \phi_0(x); \)
2. \( f(x_t) \leq \phi^*_t \triangleq \min_{x \in \mathbb{R}^p} \phi_t(x). \)

How to build an estimate sequence?

Define \( \phi_t \) recursively

\[
\phi_t(x) \triangleq (1 - \alpha_t)\phi_{t-1}(x) + \alpha_t d_t(x),
\]

where \( d_t \) is a lower-bound, e.g., if \( f \) is smooth,

\[
d_t(x) \triangleq f(y_t) + \nabla f(y_t)\top (x - y_t) + \frac{\mu}{2} \|x - y_t\|^2_2,
\]

Then, work hard to choose \( \alpha_t \) as large as possible, and \( y_t \) and \( x_t \) such that property 2 holds. Subsequently, \( \lambda_t = \prod_{t=1}^{t}(1 - \alpha_t). \)
Part II: Stochastic optimization and variance reduction
Stochastic optimization

Figure: Adaline, [Widrow and Hoff, 1960]: A physical device that performs least square regression using stochastic gradient descent.
Problems considered in this part

Minimization of (large) finite sums

\[
\min_{x \in \mathbb{R}^p} \left\{ f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \psi(x) \right\}.
\]

Minimization of expectations with infinite data

\[
\min_{x \in \mathbb{R}^p} \left\{ f(x) = \mathbb{E}_z [\ell(x, z)] + \psi(x) \right\}.
\]

The finite-sum problem corresponds to the empirical risk minimization problem, whereas the second one corresponds to the expected cost.
The stochastic gradient descent algorithm

Consider now the minimization of an expectation

$$\min_{x \in \mathbb{R}^p} f(x) = \mathbb{E}_z[\ell(x, z)],$$

To simplify, we assume that for all $z$, $x \mapsto \ell(x, z)$ is differentiable.

Algorithm

At iteration $t$,

- Randomly draw one example $z_t$ from the training set;
- Update the current iterate

$$x_t \leftarrow x_{t-1} - \eta_t \nabla f_t(x_{t-1}) \quad \text{with} \quad f_t(x) = \ell(x, z_t).$$

- Perform online averaging of the iterates (optional)

$$\tilde{x}_t \leftarrow (1 - \gamma_t)\tilde{x}_{t-1} + \gamma_t x_t.$$
The stochastic gradient descent algorithm

There are various learning rates strategies (constant, varying step-sizes), and averaging strategies. Depending on the problem assumptions and choice of $\eta_t$, $\gamma_t$, classical convergence rates may be obtained:

- $f(\tilde{x}_t) - f^* = O(1/\sqrt{t})$ for convex problems;
- $f(\tilde{x}_t) - f^* = O(1/t)$ for strongly-convex ones;

Remarks

- The convergence rates are not great, but the complexity per-iteration is small (1 gradient evaluation for minimizing an empirical risk versus $n$ for the batch algorithm).
- When the amount of data is infinite, the method minimizes the expected risk (which is what we want).
- Due to Robbins and Monro [1951].

[Nemirovski, Juditsky, Lan, and Shapiro, 2009, Moulines and Bach, 2011]...
The stochastic gradient descent algorithm

What theory tells us

- first use a **constant step-size**: the objective function value decreases quickly (as full GD) until it oscillates.
- then, use a **decreasing step size** and start **averaging**.
The stochastic gradient descent algorithm

What theory tells us

- first use a **constant step-size**: the objective function value decreases quickly (as full GD) until it oscillates.
- then, use a **decreasing step size** and start **averaging**.

What practice “seems” to tell us

- for deep networks, reducing twice the learning rate by 10 every $x$ epochs seems ok.
- use a mini batch (cheap parallelization), but not too large?
- use Nesterov/Heavy-ball’s extrapolation?
- use an adaptive learning rate strategy? (see next slide)
- averaging? or not?
- solutions tend to have small norm: implicit regularization?
The stochastic gradient descent algorithm

What theory tells us

- first use a **constant step-size**: the objective function value decreases quickly (as full GD) until it oscillates.
- then, use a **decreasing step size** and start **averaging**.

What practice “seems” to tell us

- for deep networks, reducing twice the learning rate by 10 every $x$ epochs seems ok.
- use a mini batch (cheap parallelization), but not too large?
- use Nesterov/Heavy-ball’s extrapolation?
- use an adaptive learning rate strategy? (see next slide)
- averaging? or not?
- solutions tend to have small norm: implicit regularization?

**Practice changes every year. Beware of big inductive claims.**
The stochastic gradient descent algorithm
Inspired by Jamie Soel’s presentation at NIPS’2018

- SGD:
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}). \]
The stochastic gradient descent algorithm
Inspired by Jamie Soel’s presentation at NIPS’2018

- SGD:
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}). \]

- Heavy-Ball momentum:
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}). \]
The stochastic gradient descent algorithm
Inspired by Jamie Soel’s presentation at NIPS’2018

- **SGD:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}). \]

- **Heavy-Ball momentum:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}). \]

- **Nesterov’s extrapolation:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1} + \beta_t (x_{t-1} - x_{t-2})) + \beta_t (x_{t-1} - x_{t-2}). \]
The stochastic gradient descent algorithm
Inspired by Jamie Soel’s presentation at NIPS’2018

- **SGD:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}). \]

- **Heavy-Ball momentum:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}). \]

- **Nesterov’s extrapolation:**
  \[ x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1} + \beta_t (x_{t-1} - x_{t-2})) + \beta_t (x_{t-1} - x_{t-2}). \]

- **AdaGrad [Duchi et al., 2011]**
  \[ x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}). \]
The stochastic gradient descent algorithm
Inspired by Jamie Soel’s presentation at NIPS’2018

- **SGD:**
  \[
  x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}).
  \]

- **Heavy-Ball momentum:**
  \[
  x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}).
  \]

- **Nesterov’s extrapolation:**
  \[
  x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1} + \beta_t (x_{t-1} - x_{t-2})) + \beta_t (x_{t-1} - x_{t-2}).
  \]

- **AdaGrad** [Duchi et al., 2011]
  \[
  x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}).
  \]

- **Adam** [Kingma and Ba, 2014]:
  \[
  x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}) + \beta_t H_t^{-1} H_{t-1} (x_{t-1} - x_{t-2}).
  \]
Back to finite sums

Consider now the case of interest:

\[
\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x),
\]

Question

Can we do as well as SGD in terms of cost per iteration, while enjoying a fast (linear) convergence rate like (accelerated or not) gradient descent?

For \( n = 1 \), no!

The rates are optimal for a “first-order local black box” [Nesterov, 2004].

For \( n \geq 1 \), yes! We need to design algorithms

- whose per-iteration computational complexity is smaller than \( n \);
- whose convergence rate may be worse than FISTA....
- ...but with a better expected computational complexity.
Incremental gradient descent methods

\[
\min_{x \in \mathbb{R}^p} \left\{ f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \right\}.
\]

Several randomized algorithms are designed with one \( \nabla f_i \) computed per iteration, with fast convergence rates, e.g., SAG [Schmidt et al., 2013]:

\[
x_k \leftarrow x_{k-1} - \frac{\gamma}{Ln} \sum_{i=1}^{n} y_i^k \quad \text{with} \quad y_i^k = \begin{cases} \nabla f_i(x_{k-1}) & \text{if } i = i_k \\ y_i^{k-1} & \text{otherwise} \end{cases}.
\]
Incremental gradient descent methods

\[
\min_{x \in \mathbb{R}^p} \left\{ f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \right\}.
\]

Several randomized algorithms are designed with one \( \nabla f_i \) computed per iteration, with fast convergence rates, e.g., SAG [Schmidt et al., 2013]:

\[
x_k \leftarrow x_{k-1} - \gamma \frac{Ln}{\sum_{i} y^k_i} \quad \text{with} \quad y^k_i = \begin{cases} 
\nabla f_i(x_{k-1}) & \text{if } i = i_k \\
 y^k_{i-1} & \text{otherwise}
\end{cases}.
\]

See also SVRG, SAGA, SDCA, MISO, Finito...

Some of these algorithms perform updates of the form

\[
x_k \leftarrow x_{k-1} - \eta_k g_k \quad \text{with} \quad \mathbb{E}[g_k] = \nabla f(x_{k-1}),
\]

but \( g_k \) has lower variance than in SGD.

[Schmidt et al., 2013, Xiao and Zhang, 2014, Defazio et al., 2014a,b, Shalev-Shwartz and Zhang, 2012, Mairal, 2015, Zhang and Xiao, 2015]
Incremental gradient descent methods

These methods achieve low (worst-case) complexity in expectation. The number of gradients evaluations to ensure $\mathbb{E}[f(x_k) - f^*] \leq \varepsilon$ is

<table>
<thead>
<tr>
<th>Method</th>
<th>$\mu &gt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISTA</td>
<td>$O \left(n \sqrt{\frac{L}{\mu}} \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
<tr>
<td>SVRG, SAG, SAGA, SDCA, MISO, Finito</td>
<td>$O \left(\max \left(n, \frac{L}{\mu}\right) \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
</tbody>
</table>
Incremental gradient descent methods

These methods achieve low (worst-case) complexity in expectation. The number of gradients evaluations to ensure $\mathbb{E}[f(x_k) - f^*] \leq \varepsilon$ is

<table>
<thead>
<tr>
<th></th>
<th>$\mu &gt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISTA</td>
<td>$O\left(n\sqrt{\frac{L}{\mu}} \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
<tr>
<td>SVRG, SAG, SAGA, SDCA, MISO, Finito</td>
<td>$O\left(\max\left(n, \frac{L}{\mu}\right) \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
</tbody>
</table>

Main features vs. stochastic gradient descent

- Same complexity per-iteration (but higher memory footprint).
- **Faster convergence** (exploit the finite-sum structure).
- **Less parameter tuning** than SGD.
- Some variants are **compatible with a composite term** $\psi$.
- SVRG is better than FISTA if $n \geq \sqrt{L/\mu}$.
Incremental gradient descent methods

These methods achieve low (worst-case) complexity in expectation. The number of gradients evaluations to ensure $\mathbb{E}[f(x_k) - f^*] \leq \varepsilon$ is

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISTA</td>
<td>$\mu &gt; 0$</td>
</tr>
<tr>
<td></td>
<td>$O\left(n \sqrt{\frac{L}{\mu} \log \left(\frac{1}{\varepsilon}\right)}\right)$</td>
</tr>
<tr>
<td>SVRG, SAG, SAGA, SDCA, MISO, Finito</td>
<td>$O\left(\max(n, \frac{\bar{L}}{\mu}) \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
</tbody>
</table>

Important remarks

- When $f_i(x) = \ell(z_i^T x)$, the memory footprint is $O(n)$ otherwise $O(dn)$, except for SVRG ($O(d)$).
- Some algorithms require an estimate of $\mu$;
- $\bar{L}$ is the average (or max) of the Lipschitz constants of the $\nabla f_i$'s.
- The $L$ for fista is the Lipschitz constant of $\nabla f$: $L \leq \bar{L}$. 
Incremental gradient descent methods
inspired from F. Bach’s slides.

Variance reduction
Consider two random variables $X, Y$ and define

$$Z = X - Y + \mathbb{E}[Y].$$

Then,

- $\mathbb{E}[Z] = \mathbb{E}[X]$ 
- $\text{Var}(Z) = \text{Var}(X) + \text{Var}(Y) - 2\text{cov}(X, Y)$.

The variance of $Z$ may be smaller if $X$ and $Y$ are positively correlated.
Incremental gradient descent methods
inspired from F. Bach’s slides.

Variance reduction
Consider two random variables $X, Y$ and define

$$Z = X - Y + \mathbb{E}[Y].$$

Then,
- $\mathbb{E}[Z] = \mathbb{E}[X]$
- $\text{Var}(Z) = \text{Var}(X) + \text{Var}(Y) - 2\text{cov}(X, Y)$.

The variance of $Z$ may be smaller if $X$ and $Y$ are positively correlated.

Why is it useful for stochastic optimization?
- step-sizes for SGD have to decrease to ensure convergence.
- with variance reduction, one may use larger constant step-sizes.
Incremental gradient descent methods

SVRG

\[ x_t = x_{t-1} - \gamma \left( \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(y) + \nabla f(y) \right), \]

where \( y \) is updated every epoch and \( \mathbb{E}[\nabla f_{i_t}(y)|\mathcal{F}_{t-1}] = \nabla f(y) \).

SAGA

\[ x_t = x_{t-1} - \gamma \left( \nabla f_{i_t}(x_{t-1}) - y_{i_t}^{t-1} + \frac{1}{n} \sum_{i=1}^{n} y_i^{t-1} \right), \]

where \( \mathbb{E}[y_{i_t}^{t-1}|\mathcal{F}_{t-1}] = \frac{1}{n} \sum_{i=1}^{n} y_i^{t-1} \) and \( y_i^t = \begin{cases} \nabla f_i(x_{t-1}) & \text{if } i = i_t \\ y_i^{t-1} & \text{otherwise} \end{cases} \).

MISO/Finito: for \( n \geq L/\mu \), same form as SAGA but

\[ \frac{1}{n} \sum_{i=1}^{n} y_i^{t-1} = -\mu x_{t-1} \quad \text{and} \quad y_i^t = \begin{cases} \nabla f_i(x_{t-1}) - \mu x_{t-1} & \text{if } i = i_t \\ y_i^{t-1} & \text{otherwise} \end{cases}. \]
Can we do even better for large finite sums?

Without vs with acceleration

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISTA</td>
<td>$\mu &gt; 0$</td>
<td>$O\left(n\sqrt{\frac{L}{\mu}} \log \left(\frac{1}{\epsilon}\right)\right)$</td>
</tr>
<tr>
<td>SVRG, SAG, SAGA, SDCA, MISO, Finito</td>
<td></td>
<td>$O\left(\max\left(n, \frac{L}{\mu}\right) \log \left(\frac{1}{\epsilon}\right)\right)$</td>
</tr>
<tr>
<td>Accelerated versions</td>
<td></td>
<td>$\tilde{O}\left(\max\left(n, \sqrt{\frac{nL}{\mu}}\right) \log \left(\frac{1}{\epsilon}\right)\right)$</td>
</tr>
</tbody>
</table>

- Acceleration for specific algorithms [Shalev-Shwartz and Zhang, 2014, Lan, 2015, Allen-Zhu, 2016].
- See [Agarwal and Bottou, 2015] for discussions about optimality.
- SVRG is better than FISTA if $n \geq \sqrt{L/\mu}$.
- AccSVRG is better than SVRG if $n \leq L/\mu$. 
Can we do even better for large finite sums?

Without vs with acceleration

<table>
<thead>
<tr>
<th></th>
<th>$\mu &gt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FISTA</td>
<td>$O\left(n\sqrt{\frac{L}{\mu}} \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
<tr>
<td>SVRG, SAG, SAGA, SDCA, MISO, Finito</td>
<td>$O\left(\max\left(n, \frac{L}{\mu}\right) \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
<tr>
<td>Accelerated versions</td>
<td>$\tilde{O}\left(\max\left(n, \sqrt{n\frac{L}{\mu}}\right) \log \left(\frac{1}{\varepsilon}\right)\right)$</td>
</tr>
</tbody>
</table>

- if $n$ is huge (one-pass learning): use SGD!
Questions about incremental methods

Do they work in practice?

- for convex objectives
  - on training error: huge improvements over well-tuned SGD.
  - on test error: less clear (not worse than SGD).
  - much easier to use than SGD.

- for non-convex objectives: nothing clear yet.

When is acceleration useful?

- when the problem is badly conditioned ($L/\mu$ large).
- when the amount of data is large, but not too large (such that one-pass un-regularized SGD does not work).
INVITED SPEAKERS
Lourdes AGAPITO - Léon BOTTOU - Kyunghyun CHO - Emmanuel DUPOUX
Martial HEBERT - Diane LARLUS - Hugo LAROCHELLE
Yann LECUN - Julien MAIRAL - Nicolas MANSARD - Rémi MUNOS
Julien PEREZ - Jean PONCE - Cordelia SCHMID - Andrew ZISSERMAN

2>6 JULY 2018
INRIA GRENOBLE

PRAIRIE Artificial Intelligence Summer School

computer vision
machine learning
natural language processing
robotics
cognitive science

Large-scale optimization for machine learning


References II


References III


References IV


