Large-Scale Optimization for Machine Learning

Julien Mairal

Inria Grenoble

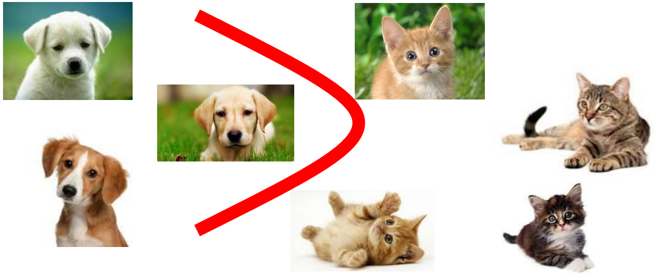
Geilo Winter School, online

Part I: Optimization is central to 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 $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathcal{X}$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$



Optimization is central to machine learning

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

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(h)}_{\text{regularization}} .$$

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 $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(h)}_{\text{regularization}} .$$

The empirical risk minimization (ERM) paradigm

1. **observe** the world (gather data);
2. **propose models** of the world (design and learn);
3. **test** on new data (estimate the generalization error).

*Very Popperian point of view, see (**??**)...*

Optimization is central to machine learning

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

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

The empirical risk minimization (ERM) paradigm, parenthesis on limitations: "("

- it is not always possible to distinguish the generalization error based on available data.
- when a complex model A performs slightly better than a simple model B, should we prefer A or B?
- we are also leaving aside the problem of non i.i.d. train/test data, biased data, testing with counterfactual reasoning... ")"

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 $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

Example 1: linear models

- 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 ℓ_2-norm $\|w\|^2$.

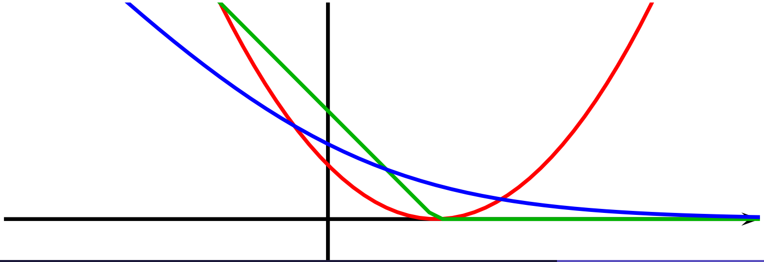
Optimization is central to machine learning

A few examples of linear models with no bias b:

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

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

Logistic regression: $$\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i w^\top x_i}\right) + \lambda \|w\|_2^2.$$



Loss as a function of $w^\top x$ with $y = 1$.

Optimization is central to machine learning

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

$$\min_{(w,b)\in\mathbb{R}^{p+1}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, w^\top x_i + b)}_{\text{empirical risk, data fit}} + \underbrace{\lambda\|w\|_2^2}_{\text{regularization}} .$$

Example 1: Why the ℓ_2-regularization for linear models $h(x) = w^\top x + b$?

- Intuition: if x and x' are similar, so should $h(x)$ and $h(x')$ be:

$$|h(x) - h(x')| \leq \|w\|_2 \|x - x'\|_2.$$

- Because we have **theory** for it (and it works in practice)!

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 $\mathcal{Y}$:

$$\min_{(w,b)\in\mathbb{R}^{p+1}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, w^\top x_i + b)}_{\text{empirical risk, data fit}} + \underbrace{\lambda\|w\|_1}_{\text{regularization}} .$$

Example 1: Why the ℓ_1-regularization for linear models $h(x) = w^\top x + b$?

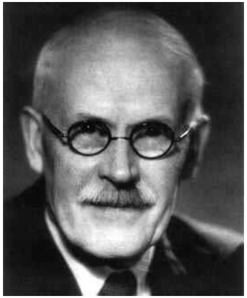
- Intuition: induces sparsity, encourages simple models.
- Because we have (too much) **theory** for it!

ℓ_1 and its variants lead to **composite optimization problems**.

(**????????**)...

Encouraging simple (sparse) models

(a) Dorothy Wrinch
1894–1980


(b) Harold Jeffreys
1891–1989

The existence of simple laws is, then, apparently, to be regarded as a quality of nature; and accordingly we may infer that it is justifiable to prefer a simple law to a more complex one that fits our observations slightly better.

(**?**). Philosophical Magazine Series.

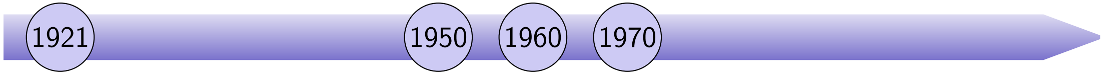
Encouraging simple (sparse) models

- 1921: Wrinch and Jeffrey's simplicity principle.

Encouraging simple (sparse) models

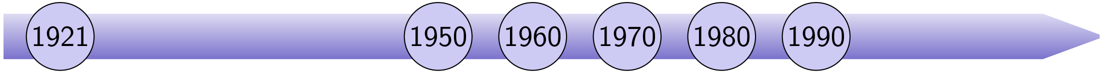
- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.

Encouraging simple (sparse) models



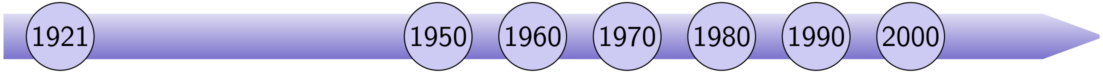
- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.

Encouraging simple (sparse) models



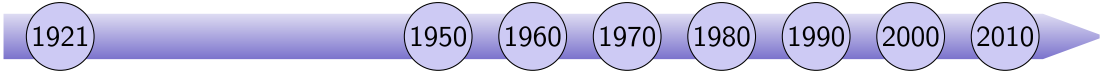
- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.
- 1990's: the **wavelet era** in signal processing.

Encouraging simple (sparse) models



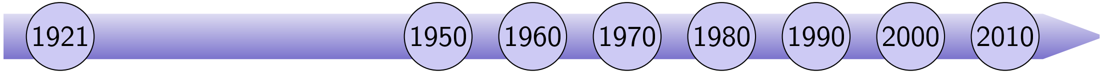
- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.
- 1990's: the **wavelet era** in signal processing.
- 1996: Olshausen and Field's **dictionary learning** method.

Encouraging simple (sparse) models



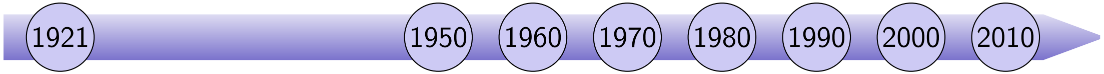
- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.
- 1990's: the **wavelet era** in signal processing.
- 1996: Olshausen and Field's **dictionary learning** method.
- 1994–1996: the **Lasso** (Tibshirani) and **Basis pursuit** (Chen and Donoho).

Encouraging simple (sparse) models



- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.
- 1990's: the **wavelet era** in signal processing.
- 1996: Olshausen and Field's **dictionary learning** method.
- 1994–1996: the **Lasso** (Tibshirani) and **Basis pursuit** (Chen and Donoho).
- 2004: **compressed sensing** (Candes, Romberg and Tao).

Encouraging simple (sparse) models

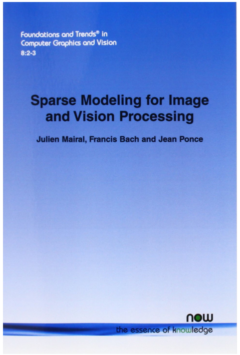


- 1921: Wrinch and Jeffrey's simplicity principle.
- 1952: Markowitz's portfolio selection.
- 1960's and 70's: best subset selection in statistics.
- 1990's: the **wavelet era** in signal processing.
- 1996: Olshausen and Field's **dictionary learning** method.
- 1994–1996: the **Lasso** (Tibshirani) and **Basis pursuit** (Chen and Donoho).
- 2004: **compressed sensing** (Candes, Romberg and Tao).
- 2006: Elad and Aharon's image denoising method.

Material on sparse estimation (free on arXiv)

long tutorial: `http://thoth.inrialpes.fr/people/mairal/resources/pdf/BigOptim.pdf`

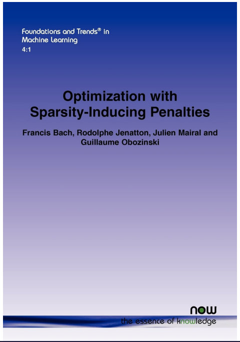
J. Mairal, F. Bach and J. Ponce. *Sparse Modeling for Image and Vision Processing*. Foundations and Trends in Computer Graphics and Vision. 2014.


Foundations and Trends® in Computer Graphics and Vision 8:2-3

Sparse Modeling for Image and Vision Processing

Julien Mairal, Francis Bach and Jean Ponce

now the essence of knowledge


Foundations and Trends® in Machine Learning 4:1

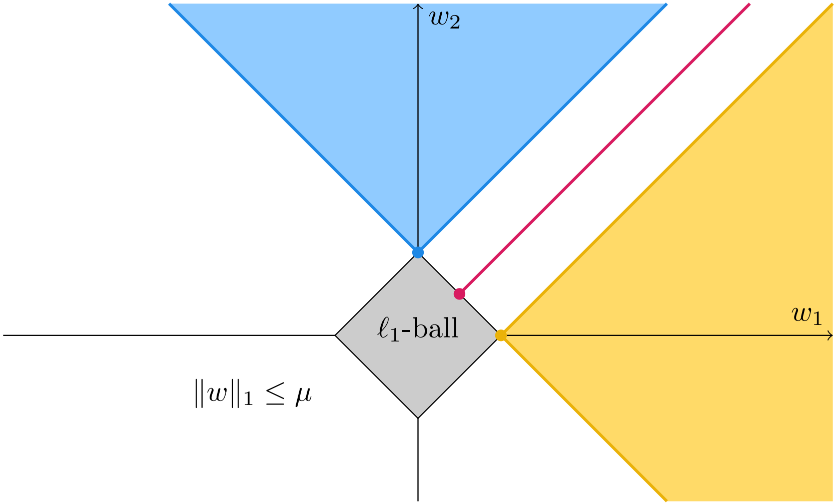
Optimization with Sparsity-Inducing Penalties

Francis Bach, Rodolphe Jenatton, Julien Mairal and Guillaume Obozinski

now the essence of knowledge

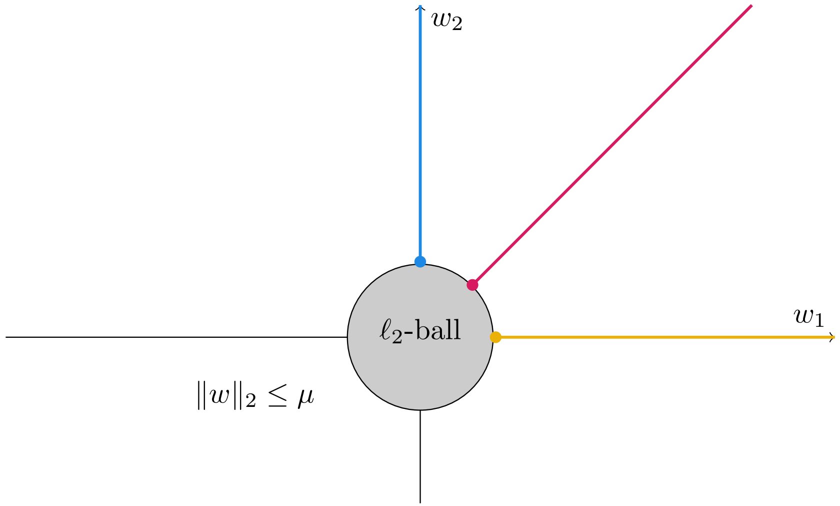
F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. *Optimization with sparsity-inducing penalties*. Foundations and Trends in Machine Learning, 4(1). 2012.

Interlude: Why does the ℓ_1-norm induce sparsity?



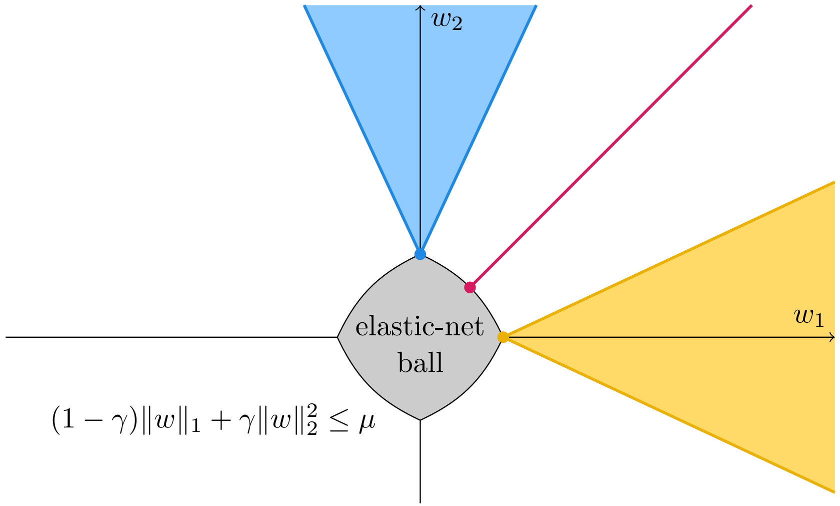
Projection onto convex sets is "biased" towards singularities.

Interlude: Why does the ℓ_1-norm induce sparsity?



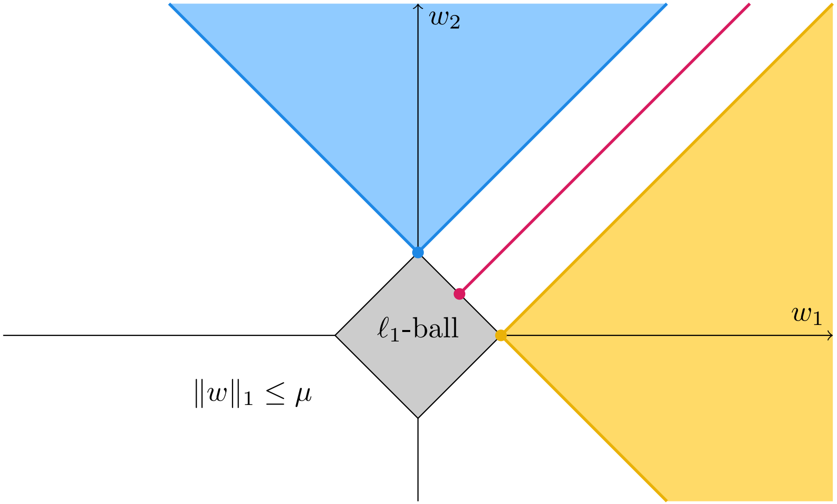
The ℓ_2-ball is isotropic.

Interlude: Why does the ℓ_1-norm induce sparsity?



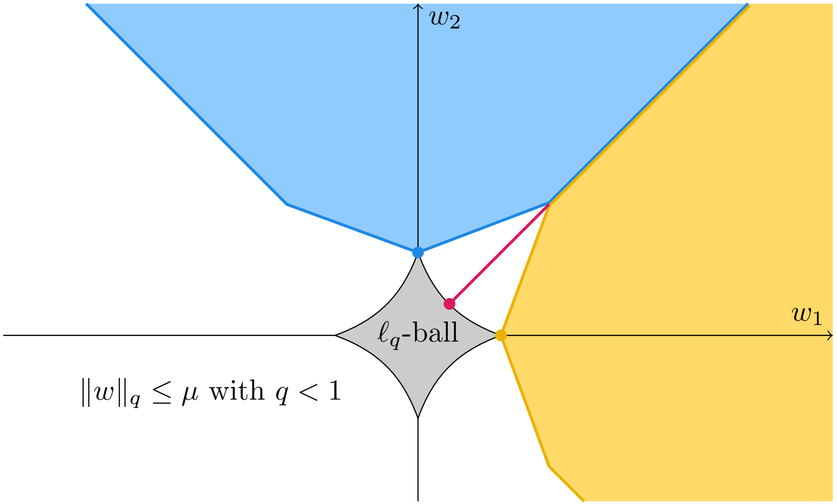
The Elastic-net penalty interpolates between ℓ_2 and ℓ_1.

Interlude: Why does the ℓ_1-norm induce sparsity?



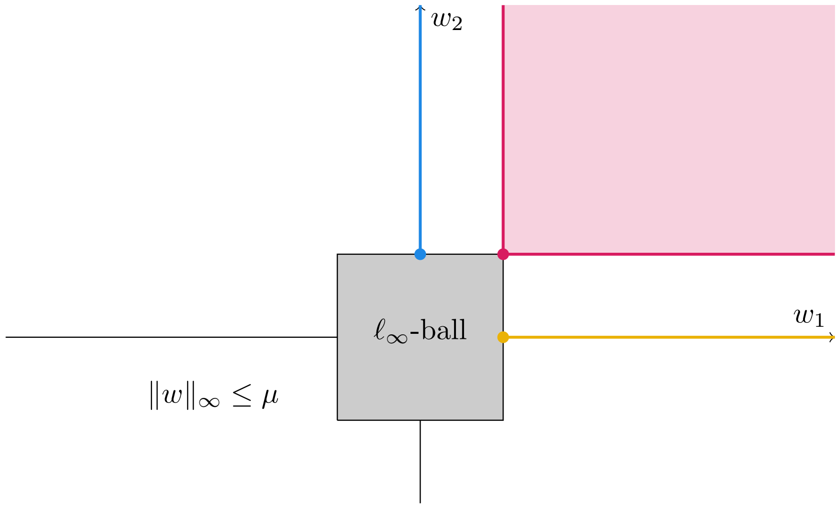
ℓ_1 again: the sparsity-inducing effect is more aggressive.

Interlude: Why does the ℓ_1-norm induce sparsity?



the sparsity-inducing effect is even more aggressive with non-convex penalties.

Interlude: Why does the ℓ_1-norm induce sparsity?



The ℓ_∞-ball encourages solutions such that $|w_1| = |w_2|$.

Optimization is central to machine learning

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

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \|h\|_{\mathcal{H}}^2}_{\text{regularization}} .$$

Example 2: kernel methods

- $\mathcal{H}$ is a **Hilbert** space (called RKHS) of functions;
- $\mathcal{H}$ and φ are **defined implicitly** through a positive definite kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$:
- Data points are mapped to the same Hilbert space through $\varphi : \mathcal{X} \rightarrow \mathcal{H}$;
- $h(x) = \langle h, \varphi(x) \rangle_{\mathcal{H}}$ is linear after mapping data to $\mathcal{H}$;

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 $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \|h\|_{\mathcal{H}}^2}_{\text{regularization}} .$$

Example 2: Why kernel methods?

- **versatility**: $\mathcal{X}$ can be anything as soon as a positive definite kernel is defined on it;
- natural way to encode **a priori knowledge** in the model (through K);
- ability to learn complex models, since $\mathcal{H}$ may be infinite-dimensional;
- **regularization is natural**: $|h(x) - h(x')| \leq \|h\|_{\mathcal{H}} \|\varphi(x) - \varphi(x')\|_{\mathcal{H}}$.

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 $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \|h\|_{\mathcal{H}}^2}_{\text{regularization}} .$$

Example 2: How do we optimize in $\mathcal{H}$?

- everything can be expressed in terms of **inner-products** $K(x_i, x_j') = \langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$;
- the solution $h^\star$ lives in the span of the $\phi(x_i)$'s: $h^\star = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$.

Optimization is central to machine learning

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

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, [\mathbf{K}\boldsymbol{\alpha}]_i)}_{\text{empirical risk, data fit}} + \underbrace{\lambda \boldsymbol{\alpha} \mathbf{K}^2 \boldsymbol{\alpha}}_{\text{regularization}} .$$

Example 2: How do we optimize in $\mathcal{H}$?

- everything can be expressed in terms of **inner-products** $K(x_i, x_j') = \langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$;
- the solution $h^\star$ lives in the span of the $\phi(x_i)$'s: $h^\star = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$.
- Then, we obtain an optimization problem (often convex) with respect to $\boldsymbol{\alpha}$ in $\mathbb{R}^n$.
- This is a 3-slides summary of a 24-hours course on kernel methods: `http://members.cbio.mines-paristech.fr/~jvert/svn/kernelcourse/slides/master2017/master2017.pdf`

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 $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathcal{X}$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(h)}_{\text{regularization}} .$$

Example 3

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 $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathcal{X}$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

Example 3

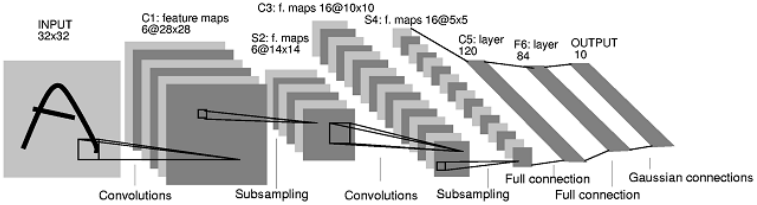
and of course, numerous contributions by other people too!

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 $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathbb{R}^p$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

Example 3: Multilayer neural networks



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 $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathbb{R}^p$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

Example 3: What is specific to multilayer neural networks?

- The "neural network" space $\mathcal{H}$ is explicitly parametrized by:

$$h(x) = \sigma_k(\mathbf{A}_k\sigma_{k-1}(\mathbf{A}_{k-1} \ldots \sigma_2(\mathbf{A}_2\sigma_1(\mathbf{A}_1 x)) \ldots)).$$

- Linear operations are either unconstrained or they share parameters (e.g., convolutions).
- Finding the optimal $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ yields a **non-convex** problem in **huge dimension.**

Optimization is central to machine learning

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

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

Example 3: What is specific to multilayer neural networks?

- The "neural network" space $\mathcal{H}$ is explicitly parametrized by:

$$h(x) = \sigma_k(\mathbf{A}_k \sigma_{k-1}(\mathbf{A}_{k-1} \ldots \sigma_2(\mathbf{A}_2 \sigma_1(\mathbf{A}_1 x)) \ldots)).$$

- Linear operations are either unconstrained or they share parameters (e.g., convolutions).
- Finding the optimal $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ yields a **non-convex** problem in **huge dimension.**

Optimization is central to machine learning

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

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

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** $h : \mathcal{X} \to \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1,\ldots,n}$ with x_i in $\mathcal{X}$, and y_i in $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda\Omega(h)}_{\text{regularization}} .$$

For over-parametrized non-convex models, optimization influences the solution!

- fitting perfectly training data is often easy with over-parametrized deep neural networks.
- . . . but **different optimization methods provide different solutions**!
- which clearly highlights new challenges for understanding the success of deep models.

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 $\mathcal{Y}$:

$$\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(h)}_{\text{regularization}} .$$

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.

Optimization is central to machine learning

Examples of unsupervised learning formulations:

$$\min_{\mathbf{D} \in \mathcal{D}} \quad \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{D}, x_i),$$

- **clustering**:

$$\mathcal{D} = \mathbb{R}^{p \times k} \quad \text{and} \quad L(\mathbf{D}, x) = \min_{j=1,\ldots,k} \|x - d_j\|^2.$$

Optimization is central to machine learning

Examples of unsupervised learning formulations:

$$\min_{\mathbf{D} \in \mathcal{D}} \quad \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{D}, x_i),$$

- **clustering**:

$$\mathcal{D} = \mathbb{R}^{p \times k} \quad \text{and} \quad L(\mathbf{D}, x) = \min_{j=1,\ldots,k} \|x - d_j\|^2.$$

- **non-negative matrix factorization** (**?**):

$$\mathcal{D} = \mathbb{R}_+^{p \times k} \quad \text{with} \quad L(\mathbf{D}, x) = \min_{\boldsymbol{\alpha} \in \mathbb{R}_+^p} \|x - \mathbf{D}\boldsymbol{\alpha}\|^2.$$

Optimization is central to machine learning

Examples of unsupervised learning formulations:

$$\min_{\mathbf{D}\in\mathcal{D}} \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{D}, x_i),$$

- **clustering**:

$$\mathcal{D} = \mathbb{R}^{p\times k} \quad \text{and} \quad L(\mathbf{D}, x) = \min_{j=1,\dots,k} \|x - d_j\|^2.$$

- **non-negative matrix factorization** (**?**):

$$\mathcal{D} = \mathbb{R}_+^{p\times k} \quad \text{with} \quad L(\mathbf{D}, x) = \min_{\boldsymbol{\alpha}\in\mathbb{R}_+^p} \|x - \mathbf{D}\boldsymbol{\alpha}\|^2.$$

- **sparse coding (dictionary learning)** (**?**):

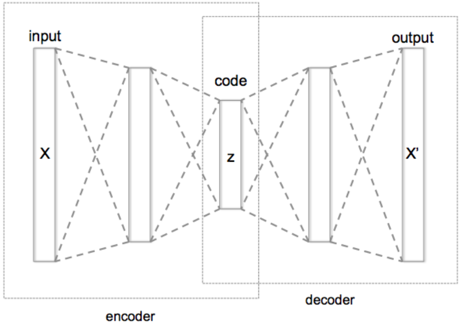
$$\mathcal{D} = \{\mathbf{D}\in\mathbb{R}^{p\times k} : \|d_j\|_2 \leq 1\} \quad \text{with} \quad L(\mathbf{D}, x) = \min_{\boldsymbol{\alpha}\in\mathbb{R}^p} \frac{1}{2}\|x - \mathbf{D}\boldsymbol{\alpha}\|^2 + \lambda\|\boldsymbol{\alpha}\|_1.$$

Optimization is central to machine learning

Examples of unsupervised learning formulations:

$$\min_{\mathbf{D}\in\mathcal{D}} \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{D}, x_i),$$

- **auto-encoders**:



Interlude: matrix factorization

Many of the previous formulations

$$\min_{\mathbf{D}\in\mathcal{D}} \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{D}, x_i) \quad \text{with} \quad L(\mathbf{D}, x) = \min_{\boldsymbol{\alpha}\in\mathcal{A}} \frac{1}{2}\|x - \mathbf{D}\boldsymbol{\alpha}\|^2 + \lambda\psi(\boldsymbol{\alpha}).$$

can be written as matrix factorization problems:

$$\min_{\mathbf{D}\in\mathcal{D},\mathbf{A}\in\mathcal{A}} \frac{1}{2}\|\mathbf{X} - \mathbf{D}\mathbf{A}\|_{\mathsf{F}}^2 + \lambda\psi(\mathbf{A}).$$

Interlude: matrix factorization

Many of the previous formulations

$$\min_{\mathbf{D}\in\mathcal{D}} \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{D}, x_i) \quad \text{with} \quad L(\mathbf{D}, x) = \min_{\boldsymbol{\alpha}\in\mathcal{A}} \frac{1}{2}\|x - \mathbf{D}\boldsymbol{\alpha}\|^2 + \lambda\psi(\boldsymbol{\alpha}).$$

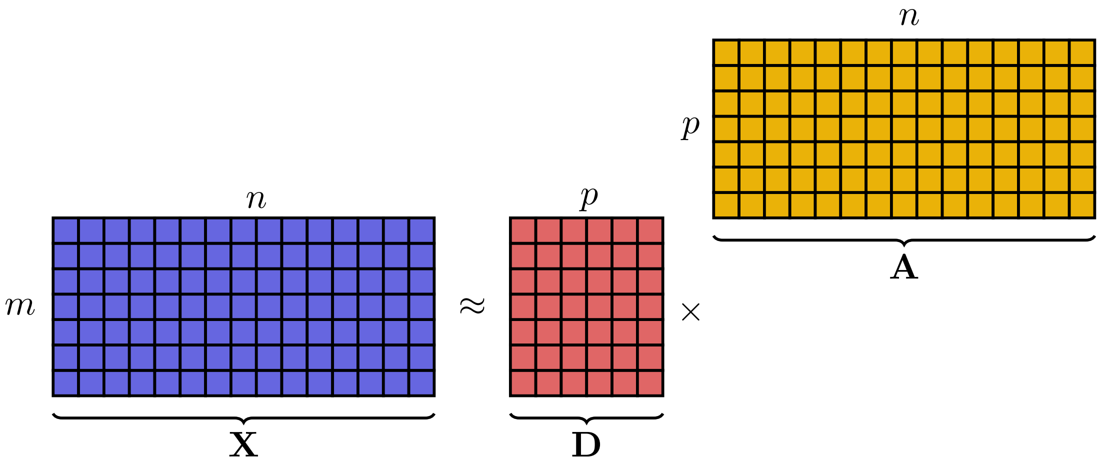
can be written as matrix factorization problems:

$$\min_{\mathbf{D}\in\mathcal{D},\mathbf{A}\in\mathcal{A}} \frac{1}{2}\|\mathbf{X} - \mathbf{D}\mathbf{A}\|_{\mathsf{F}}^2 + \lambda\psi(\mathbf{A}).$$

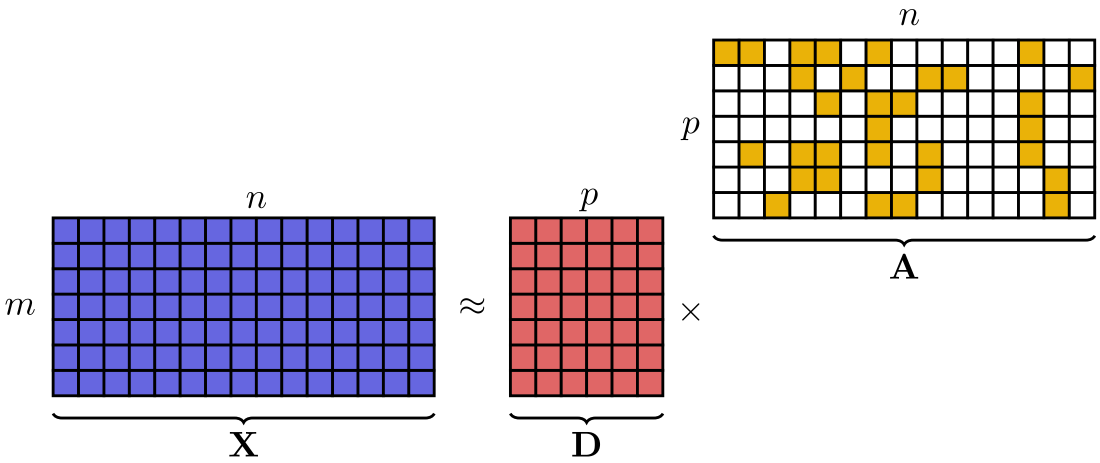
which is a key technique for unsupervised data modeling

- recommender systems (Netflix prize) and social networks.
- document clustering.
- genomic pattern discovery.
- image processing. . .

Interlude: matrix factorization

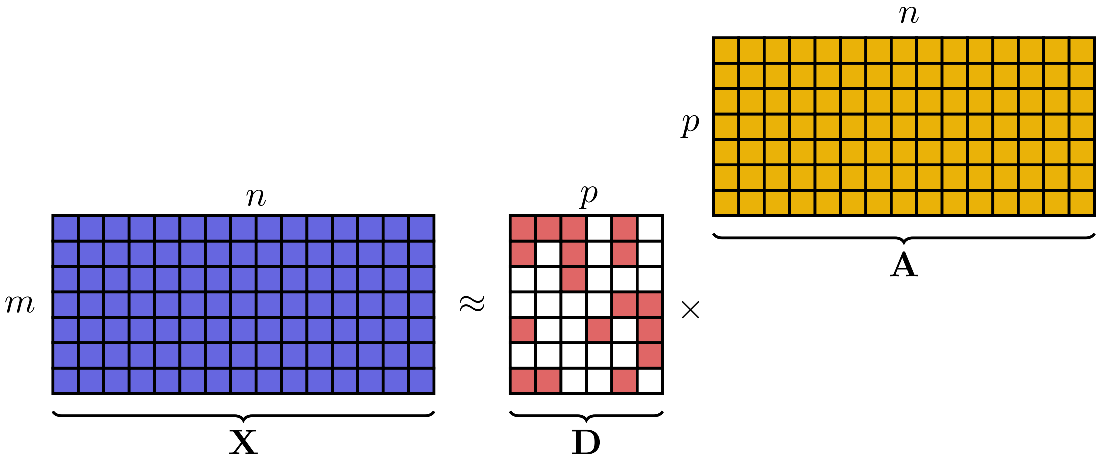


Interlude: matrix factorization



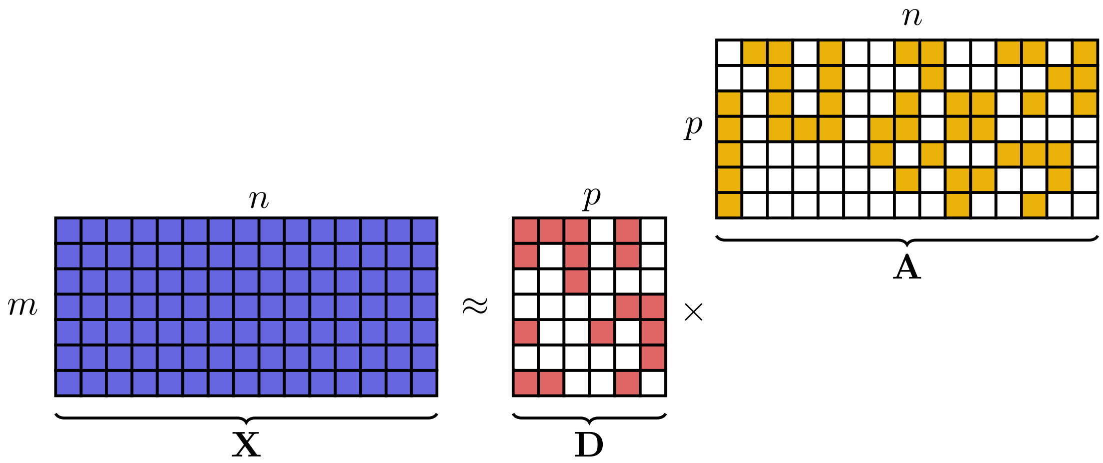
when a factor is **sparse**.

Interlude: matrix factorization



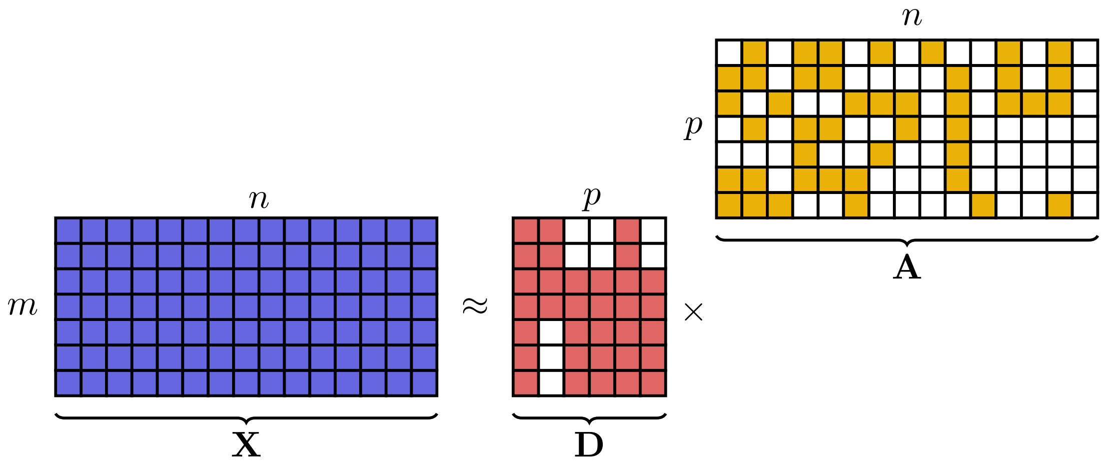
or the other one.

Interlude: matrix factorization



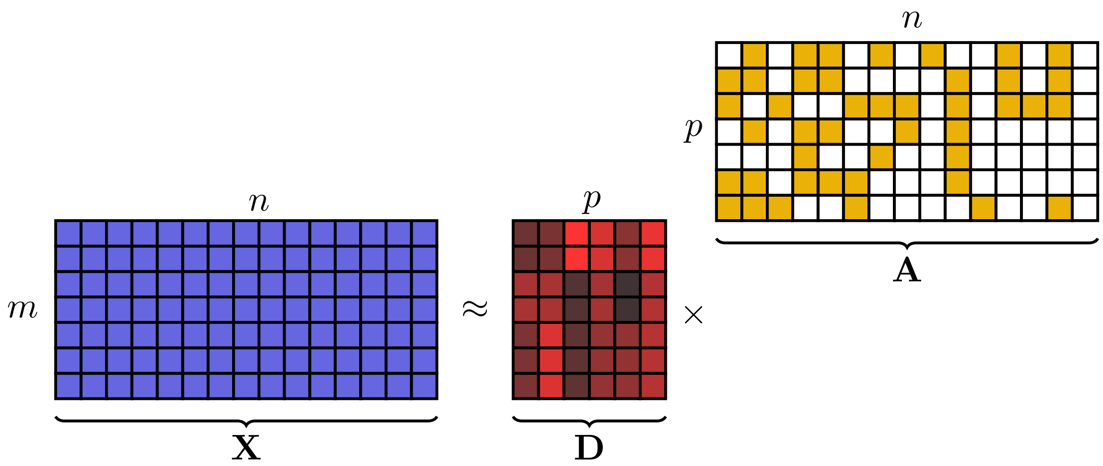
or both.

Interlude: matrix factorization



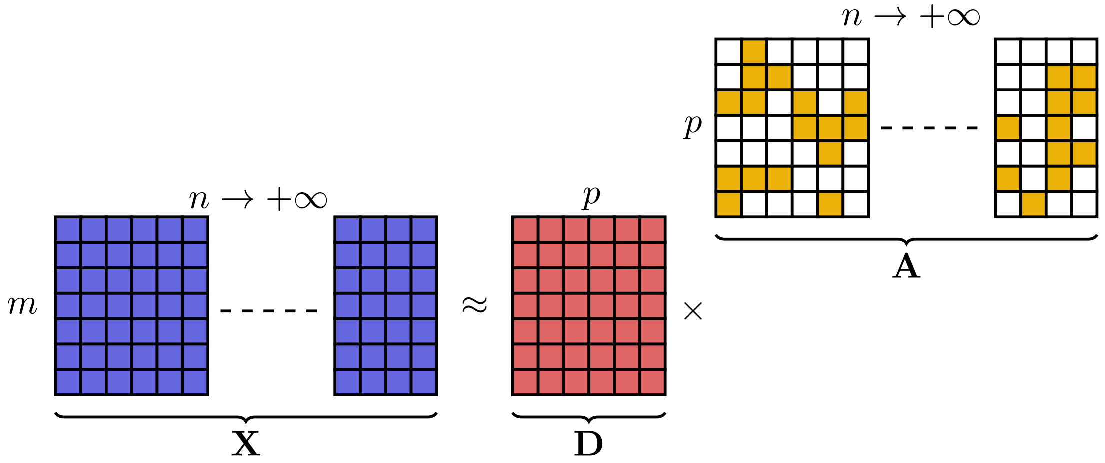
or not only one factor is **sparse**, but it admits a **particular structure**.

Interlude: matrix factorization



or one factor admits a **particular structure** (*e.g.*, piecewise constant), but it is not sparse.

Interlude: matrix factorization

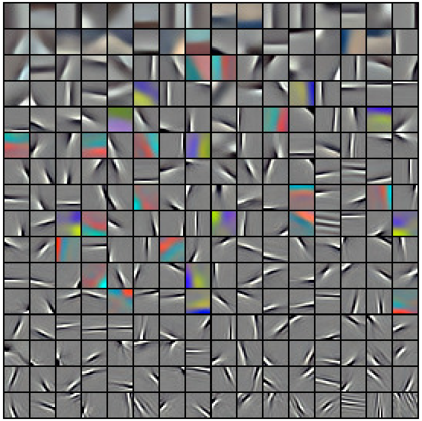


or the matrix admits an **infinite number of columns**, or columns are **streamed online**.

Interlude: The sparse coding formulation

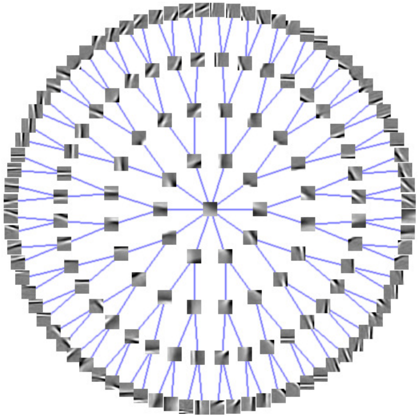
was introduced by Olshausen and Field, '96. It was the first time (together with ICA, see [Bell and Sejnowski, '97]) that a **simple unsupervised learning principle** would lead to

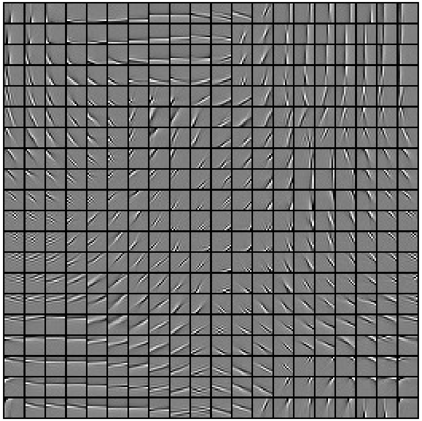
various sorts of "Gabor-like" filters, when trained on natural image patches.



Interlude: The sparse coding formulation

or with other **structured sparsity-inducing penalties**:





[Jenatton et al. 2010], [Kavukcuoglu et al., 2009], [Mairal et al. 2011], [Hyvärinen and Hoyer, 2001].

Interlude: The archetypal analysis formulation

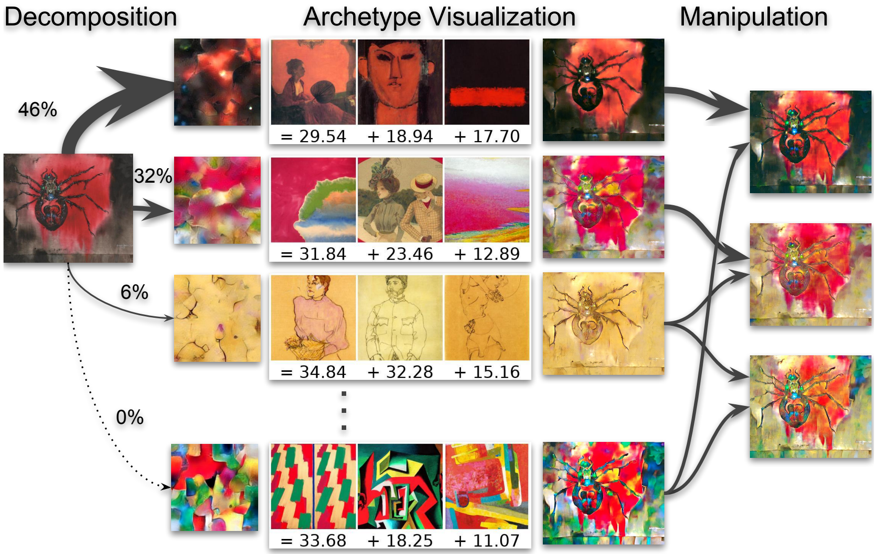
$$\min_{\mathbf{B}\in\mathcal{B},\mathbf{A}\in\mathcal{A}} \frac{1}{2}\|\mathbf{X}-\mathbf{DA}\|_{\mathsf{F}}^2 \quad \text{s.t.} \quad \mathbf{D}=\mathbf{XB},$$

The columns of $\mathbf{A}$ and $\mathbf{B}$ are constrained to be in the simplex.

- archetypes are convex combinations of data points.
- data points are close to convex combinations of arechetypes.

[Cutler and Breiman, 1994].

Interlude: archetypal analysis for style representation [Dwynen et al., 2018].



Large-scale optimization for machine learning

What would be a great outline for this tutorial

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

Large-scale optimization for machine learning

What we will do

- Introduction to statistical learning and gradient-based optimization.
- Introduction to **stochastic** optimization.
- Two or three advanced topics:
  - **Variance-reduced stochastic gradient descent.**
  - **Nesterov's acceleration (momentum)**.

Large-scale optimization for machine learning

What we will do

- Introduction to statistical learning and gradient-based optimization.
- Introduction to **stochastic** optimization.
- Two or three advanced topics:
  - **Variance-reduced stochastic gradient descent.**
  - **Nesterov's acceleration (momentum)**.

What does "large-scale" mean?

In this tutorial, it means a problem that fits into a big computer's main memory ($\leq$ 1TB).

Part II: 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

1. 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:

$$R(\hat{h}_n) - \min_h R(h) = \underbrace{R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h)}_{\text{estimation error}} + \underbrace{\min_{h \in \mathcal{H}} R(h) - \min_h R(h)}_{\text{approximation error}}$$

- 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:

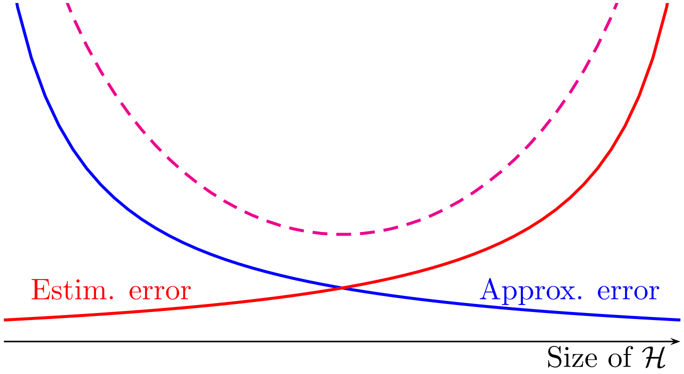
$$R(\hat{h}_n) - \min_h R(h) = \underbrace{R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h)}_{\text{estimation error}} + \underbrace{\min_{h \in \mathcal{H}} R(h) - \min_h R(h)}_{\text{approximation error}}$$

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

$$R(\tilde{h}_n) - \min_h R(h) = \underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{optimization error}} + R(\hat{h}_n) - \min_h R(h)$$

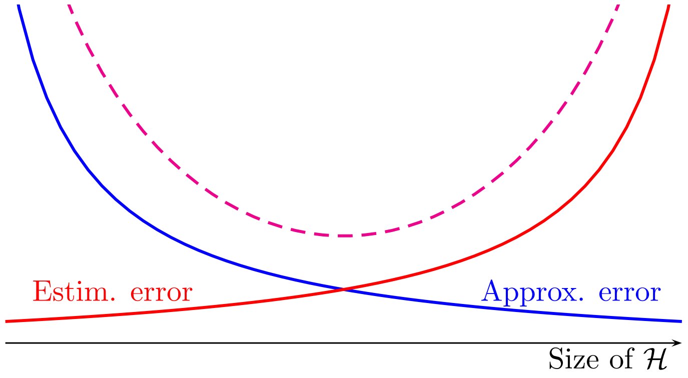
- Insight of Bottou and Bousquet (2008): **no need to optimize below statistical error!**

Statistical learning



- Illustration of the Approximation/Estimation trade-off without considering optimization cost, inspired from L. Bottou's tutorial.

Statistical learning



- Illustration of the Approximation/Estimation trade-off without considering optimization cost, inspired from L. Bottou's tutorial.
- ...but when optimization comes into play, things become more complicated, especially when the optimization algorithm influences the approximation 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 $O(1/n)$.

more details in `http://www.di.ens.fr/~fbach/fbach_frejus_2017.pdf`

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 $O(1/n)$.

more details in `http://www.di.ens.fr/~fbach/fbach_frejus_2017.pdf`

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

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

more details in `http://www.di.ens.fr/~fbach/fbach_frejus_2017.pdf`

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, except 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 $O(1/n)$.

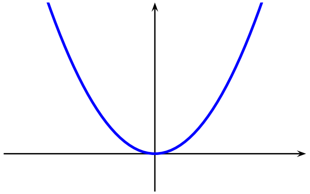
more details in `http://www.di.ens.fr/~fbach/fbach_frejus_2017.pdf`

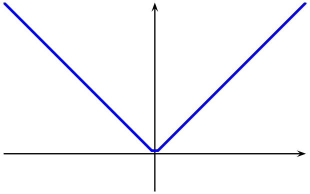
What conclusions can we draw from an optimization perspective?

- convergence rate of stochastic gradient descent (at least for convex problems) may be **asymptotically optimal**.
- faster algorithms than SGD are not always useful, except 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.**

Basics of gradient-based optimization

Smooth vs non-smooth


(a) smooth

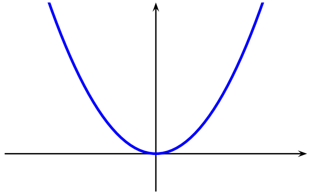

(b) non-smooth

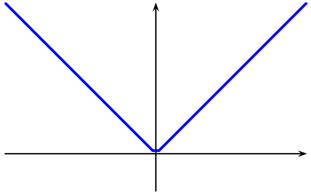
An important quantity to quantify smoothness is the **Lipschitz constant** of the gradient:

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

Basics of gradient-based optimization

Smooth vs non-smooth


(a) smooth


(b) non-smooth

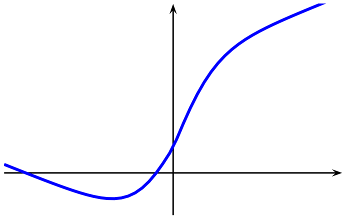
An important quantity to quantify smoothness is the **Lipschitz constant** of the gradient:

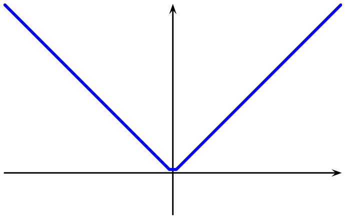
$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

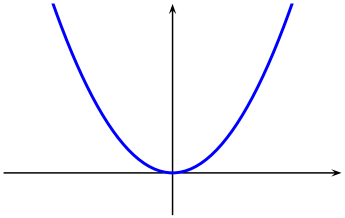
If f is twice differentiable, L may be chosen as the **largest eigenvalue** of the Hessian $\nabla^2 f$. This is an upper-bound on the function curvature.

Basics of gradient-based optimization

Convex vs non-convex


(a) non-convex


(b) convex

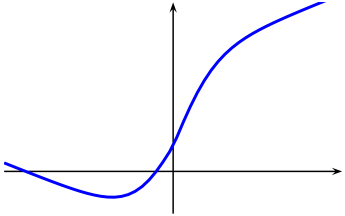

(c) strongly-convex

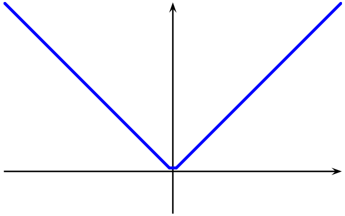
An important quantity to quantify convexity is the **strong-convexity** constant

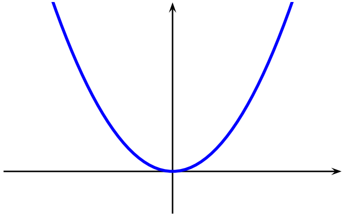
$$f(x) \geq f(y) + \nabla f(y)^\top (x-y) + \frac{\mu}{2}\|x-y\|^2,$$

Basics of gradient-based optimization

Convex vs non-convex


(a) non-convex


(b) convex


(c) strongly-convex

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

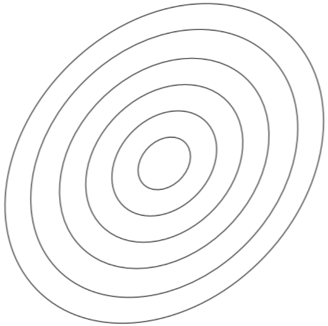
$$f(x) \geq f(y) + \nabla f(y)^\top (x - y) + \frac{\mu}{2} \|x - y\|^2,$$

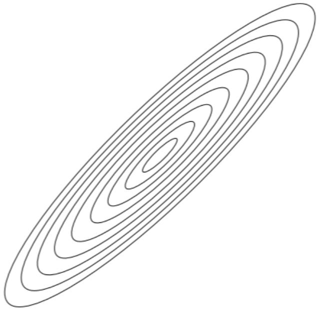
If f is twice differentiable, μ may be chosen as the **smallest eigenvalue** of the Hessian $\nabla^2 f$. This is a lower-bound on the function curvature.

Basics of gradient-based optimization

Picture from F. Bach

Why is the condition number L/μ important?

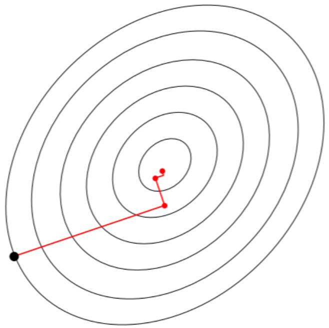

(small $\kappa = L/\mu$)


(large $\kappa = L/\mu$)

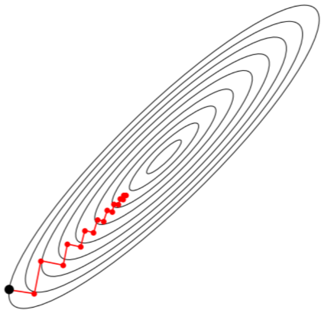
Basics of gradient-based optimization

Picture from F. Bach

Trajectory of gradient descent with optimal step size.



(small $\kappa = L/\mu$)

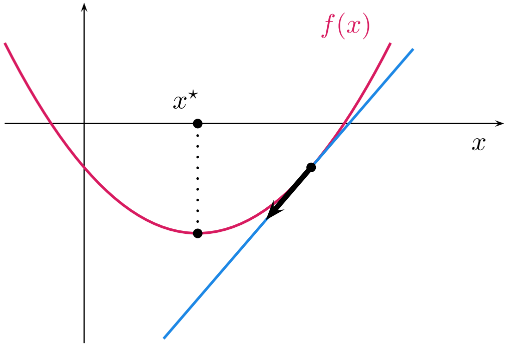


(large $\kappa = L/\mu$)

Basics of gradient-based optimization

Convex Functions

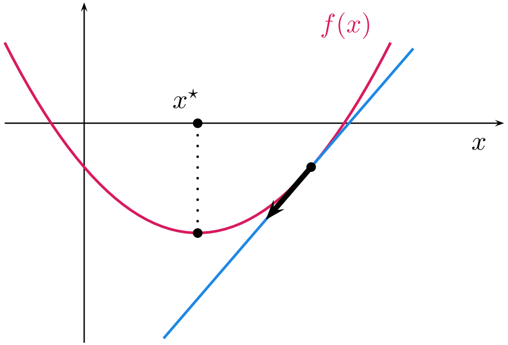
Why do we care about convexity?



Basics of gradient-based optimization

Convex Functions

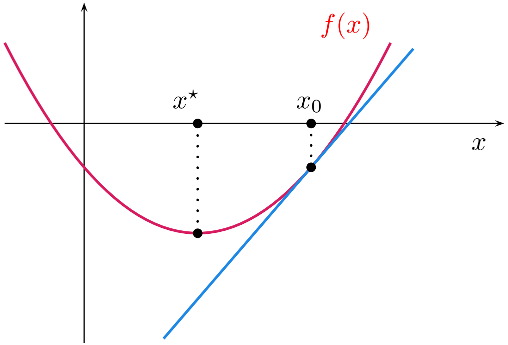
Local observations give information about the global optimum



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

Basics of gradient-based optimization

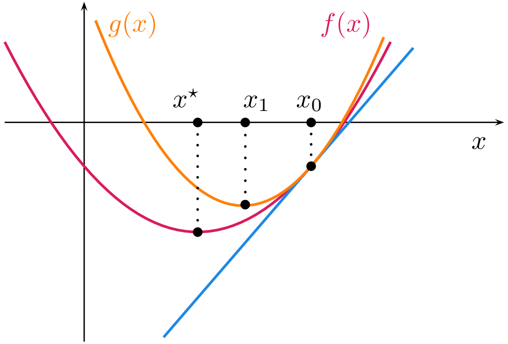
If f is convex and smooth



- $f(x) \geq \underbrace{f(x_0) + \nabla f(x_0)^\top (x - x_0)}_{\text{linear approximation}}$;
- if f is non-smooth, a similar inequality holds for subgradients.

Basics of gradient-based optimization

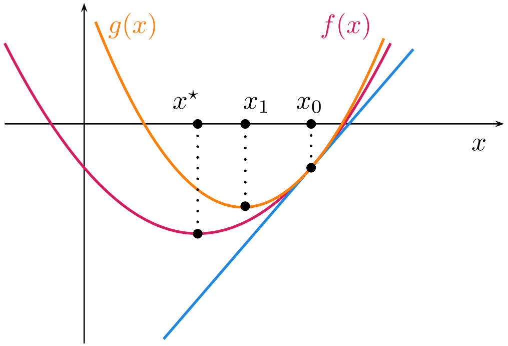
If ∇f is L-Lipschitz continuous (f does not need to be convex)



- $f(x) \leq g(x) = \underbrace{f(x_0) + \nabla f(x_0)^\top (x - x_0)}_{\text{linear approximation}} + \frac{L}{2}\|x - x_0\|_2^2$;

Basics of gradient-based optimization

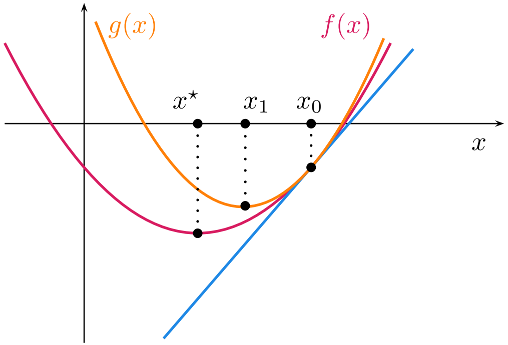
If ∇f is L-Lipschitz continuous (f does not need to be convex)



- $f(x) \leq g(x) = \underbrace{f(x_0) + \nabla f(x_0)^\top (x - x_0)}_{\text{linear approximation}} + \frac{L}{2}\|x - x_0\|_2^2$;
- $g(x) = C_{x_0} + \frac{L}{2}\|x_0 - (1/L)\nabla f(x_0) - x\|_2^2$.

Basics of gradient-based optimization

If ∇f is L-Lipschitz continuous (f does not need to be convex)



- $f(x) \leq g(x) = \underbrace{f(x_0) + \nabla f(x_0)^\top (x - x_0)}_{\text{linear approximation}} + \frac{L}{2}\|x - x_0\|_2^2$;
- $\boxed{x_1 = x_0 - \frac{1}{L}\nabla f(x_0) \text{ (gradient descent step).}}$

Basics of gradient-based optimization

Gradient descent algorithm

Assume that f is convex and L-smooth (∇f is L-Lipschitz).

Theorem

Consider the algorithm

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

Then,

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

Basics of gradient-based optimization

Gradient descent algorithm

Assume that f is convex and L-smooth (∇f is L-Lipschitz).

Theorem

Consider the algorithm

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

Then,

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

Complexity point of view

To guarantee $f(x_t) - f^\star \leq \varepsilon$, we need $O(L/\varepsilon)$ iterations.

Basics of gradient-based optimization

Gradient descent algorithm

Assume that f is convex and L-smooth (∇f is L-Lipschitz).

Theorem

Consider the algorithm

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

Then,

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

How to prove this?

Read Nesterov's book! (?).

Proof (1/2)

Proof of the main inequality for smooth functions

We want to show that for all x and z,

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

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,

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

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,

$$\begin{aligned} f(x) - f(z) - \nabla f(z)^\top (x - z) &= \int_0^1 (\nabla f(tx + (1-t)z) - \nabla f(z))^\top (x - z) dt \\ &\leq \int_0^1 |(\nabla f(tx + (1-t)z) - \nabla f(z))^\top (x - z)| dt \\ &\leq \int_0^1 \|\nabla f(tx + (1-t)z) - \nabla f(z)\|_2 \|x - z\|_2 dt \quad \text{(C.-S.)} \\ &\leq \int_0^1 Lt\|x - z\|_2^2 dt = \frac{L}{2}\|x - z\|_2^2. \end{aligned}$$

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,

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,

$$\mathbf{f(x_t) \leq g_t(x_t)}$$

Proof (2/2)

Proof of the theorem

We have shown that for all x,

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

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,

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

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,

$$\begin{aligned} f(x_t) &\leq g_t(x_t) = \mathbf{g_t(x^\star)} - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &= \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1}) + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2} - \frac{L}{2}\|x^\star - x_t\|_2^2 \end{aligned}$$

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,

$$\begin{aligned} f(x_t) &\leq g_t(x_t) = g_t(x^\star) - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &= \mathbf{f}(\mathbf{x_{t-1}}) + \nabla \mathbf{f}(\mathbf{x_{t-1}})^\top (\mathbf{x}^\star - \mathbf{x_{t-1}}) + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &\leq \mathbf{f}^\star + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2. \end{aligned}$$

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,

$$\begin{aligned} f(x_t) \leq g_t(x_t) &= g_t(x^\star) - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &= \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1})} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &\leq \mathbf{f}^\star + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2. \end{aligned}$$

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

$$T(f(x_T) - f^\star) \leq \sum_{t=1}^{T} f(x_t) - f^\star \leq \frac{L}{2}\|x^\star - x^0\|_2^2 - \frac{L}{2}\|x^\star - x_T\|_2^2 \leq \frac{L}{2}\|x^\star - x^0\|_2^2.$$

Proof (2/2)

Proof of the theorem

We have shown that for all x,

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

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,

$$\begin{aligned} f(x_t) \leq g_t(x_t) &= g_t(x^\star) - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &= \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1})} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &\leq \mathbf{f^\star} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2. \end{aligned}$$

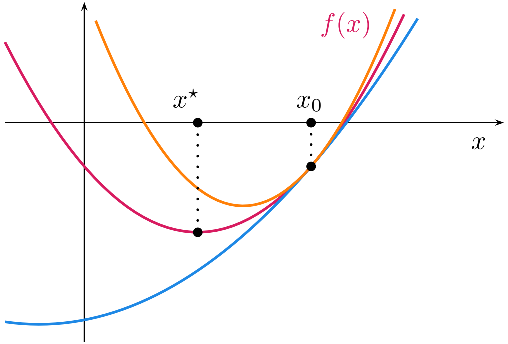
By summing from $t = 1$ to T, we have a telescopic sum

$$T(f(x_T) - f^\star) \leq \sum_{t=1}^{T} f(x_t) - f^\star \leq \frac{L}{2}\|x^\star - x^0\|_2^2 - \frac{L}{2}\|x^\star - x_T\|_2^2 \leq \frac{L}{2}\|x^\star - x^0\|_2^2.$$

(orange) - (red) - (blue) - telescopic sum

Basics of gradient-based optimization

If ∇f is L-Lipschitz continuous and f μ-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 μ-strongly convex and L-smooth, the gradient descent algorithm with step-size $1/L$ produces iterates such that

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

We call that a **linear** convergence rate.

Remarks

- if f is twice differentiable, L and μ represent the larget and smallest eigenvalues of the Hessian, respectively.
- L/μ is called the **condition number**.

Basics of gradient-based optimization

Proposition

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

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

We call that a **linear** convergence rate.

Complexity point of view

The number of iterations to guarantee $f(x_t) - f^\star \leq \varepsilon$ is upper bounded by

$$O\left(\frac{L}{\mu} \log\left(\frac{L\|x_0 - x^\star\|^2}{\varepsilon}\right)\right).$$

Proof

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

$$
\begin{aligned}
f(x_t) &\leq \mathbf{f}(\mathbf{x_{t-1}}) + \nabla \mathbf{f}(\mathbf{x_{t-1}})^\top (\mathbf{x}^\star - \mathbf{x_{t-1}}) + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\
&\leq \mathbf{f}^\star + \frac{L - \boldsymbol{\mu}}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2.
\end{aligned}
$$

Proof

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

$$\begin{aligned} f(x_t) &\leq \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1})} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &\leq \mathbf{f^\star} + \frac{L - \boldsymbol{\mu}}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2. \end{aligned}$$

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

$$\begin{aligned} \|x^\star - x_t\|_2^2 &\leq \frac{L-\mu}{L+\mu}\|x^\star - x_{t-1}\|_2^2 \\ &\leq \left(1 - \frac{\mu}{L}\right)\|x^\star - x_{t-1}\|_2^2 \leq \left(1 - \frac{\mu}{L}\right)^t \|x^\star - x_0\|^2. \end{aligned}$$

Proof

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

$$
\begin{aligned}
f(x_t) &\leq \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1})} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\
&\leq \mathbf{f}^\star + \frac{L-\boldsymbol{\mu}}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2.
\end{aligned}
$$

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

$$
\begin{aligned}
\|x^\star - x_t\|_2^2 &\leq \frac{L-\mu}{L+\mu}\|x^\star - x_{t-1}\|_2^2 \\
&\leq \left(1 - \frac{\mu}{L}\right)\|x^\star - x_{t-1}\|_2^2 \leq \left(1 - \frac{\mu}{L}\right)^t \|x^\star - x_0\|^2.
\end{aligned}
$$

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

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

Proof

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

$$\begin{aligned} f(x_t) &\leq \mathbf{f(x_{t-1}) + \nabla f(x_{t-1})^\top (x^\star - x_{t-1})} + \frac{L}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2 \\ &\leq \mathbf{f^\star} + \frac{L - \boldsymbol{\mu}}{2}\|x^\star - x_{t-1}\|_2^2 - \frac{L}{2}\|x^\star - x_t\|_2^2. \end{aligned}$$

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

$$\begin{aligned} \|x^\star - x_t\|_2^2 &\leq \frac{L-\mu}{L+\mu}\|x^\star - x_{t-1}\|_2^2 \\ &\leq \left(1 - \frac{\mu}{L}\right)\|x^\star - x_{t-1}\|_2^2 \leq \left(1 - \frac{\mu}{L}\right)^t \|x^\star - x_0\|^2. \end{aligned}$$

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

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

It is all about orange 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

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

where f_0 is L-smooth and ψ is convex but not necessarily smooth.

Examples

- ℓ_1-norm: $\psi(x) = \|x\|_1$, 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

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

where f_0 is L-smooth and ψ is convex but not necessarily smooth.

Examples

- ℓ_1-norm: $\psi(x) = \|x\|_1$, which induces sparsity;
- Group Lasso: $\psi(x) = \sum_{g \in \mathcal{G}} \|x[g]\|_2$;

Basics of gradient-based optimization: composite problems

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

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

where f_0 is L-smooth and ψ is convex but not necessarily smooth.

Examples

- ℓ_1-norm: $\psi(x) = \|x\|_1$, which induces sparsity;
- Group Lasso: $\psi(x) = \sum_{g \in \mathcal{G}} \|x[g]\|_2$;
- Total variation $\psi(x) = \sum_{i=2}^{p} |x[i] - x[i-1]|$ (here in 1D);

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 ψ is convex but not necessarily smooth.

Examples

- ℓ_1-norm: $\psi(x) = \|x\|_1$, which induces sparsity;
- Group Lasso: $\psi(x) = \sum_{g \in \mathcal{G}} \|x[g]\|_2$;
- Total variation $\psi(x) = \sum_{i=2}^{p} |x[i] - x[i-1]|$ (here in 1D);
- Indicator function of a convex set

$$\psi(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ +\infty & \text{otherwise.} \end{cases}$$

Basics of gradient-based optimization: composite problems

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

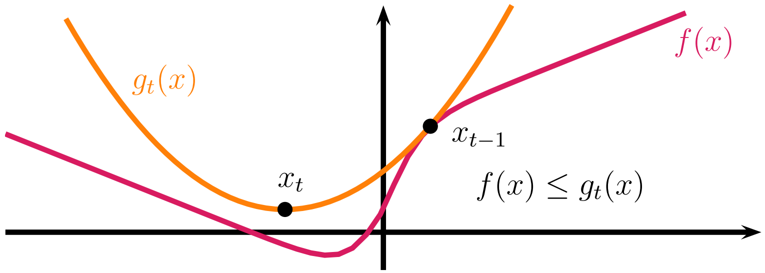
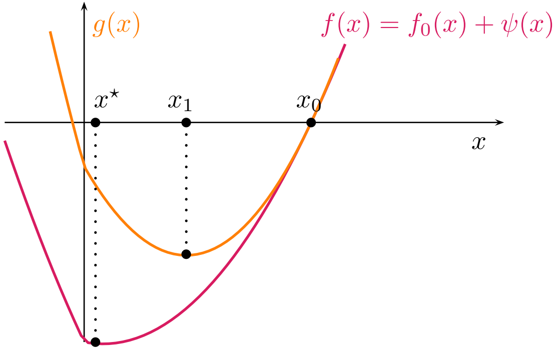


Figure: At each step, we update $x_t \in \arg\min_{x \in \mathbb{R}^p} g_t(x)$

Basics of gradient-based optimization: composite problems

An important inequality for composite functions

If ∇f_0 is L-Lipschitz continuous

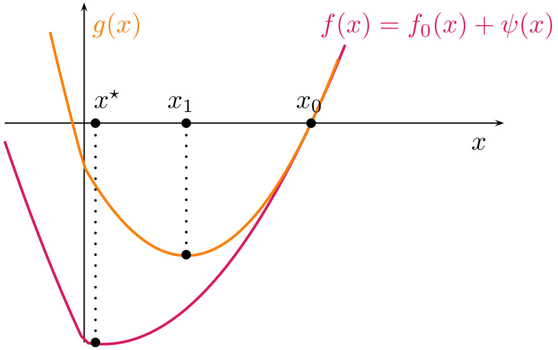


- $f_0(x) \leq f_0(x_0) + \nabla f_0(x_0)^\top (x - x_0) + \frac{L}{2}\|x - x_0\|_2^2$;

Basics of gradient-based optimization: composite problems

An important inequality for composite functions

If ∇f_0 is L-Lipschitz continuous



- $f_0(x) + \psi(x) \leq f_0(x_0) + \nabla f_0(x_0)^\top (x - x_0) + \frac{L}{2}\|x - x_0\|_2^2 + \psi(x)$;
- x_1 minimizes g.

Basics of gradient-based optimization: composite problems

Gradient descent for minimizing f consists of

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

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

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

It requires computing efficiently the **proximal operator** (**?**) of ψ.

$$y \mapsto \arg\min_{x \in \mathbb{R}^p} \ \frac{1}{2}\|y - x\|_2^2 + \psi(x).$$

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 ℓ_1

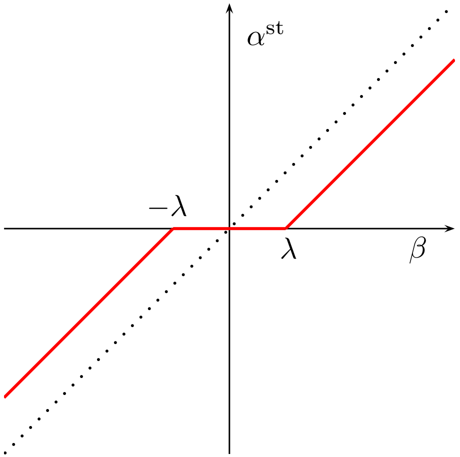
The proximal operator of $\lambda\|.\|_1$ is the soft-thresholding operator

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

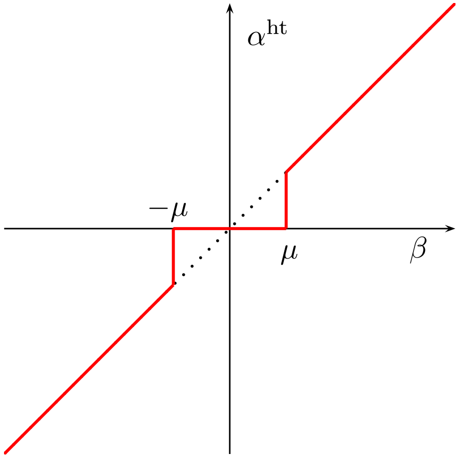
The resulting algorithm is called **iterative soft-thresholding**.

(**???????**)...

Basics of gradient-based optimization: composite problems



(a) Soft-thresholding operator, $\alpha^{\mathsf{st}} = \mathsf{sign}(\beta) \max(|\beta| - \lambda, 0)$.



(b) Hard-thresholding operator$\alpha^{\mathsf{ht}} = \delta_{|\beta| \geq \mu} \beta$.

Basics of gradient-based optimization: composite problems

Proximal operator of ℓ_1:

$$\min_{x \in \mathbb{R}} \frac{1}{2}(y - x)^2 + \lambda |x|$$

Piecewise quadratic function with a kink at zero.

Derivative at 0_+: $g_+ = -y + \lambda$ and 0_-: $g_- = -y - \lambda$.

Optimality conditions. x is optimal iff:

- $|x| > 0$ and $(y - x) + \lambda \operatorname{sign}(x) = 0$
- $x = 0$ and $g_+ \geq 0$ and $g_- \leq 0$

The solution is a **soft-thresholding**:

$$x^\star = \operatorname{sign}(y)(|y| - \lambda)^+.$$

Basics of gradient-based optimization: composite problems

Proximal operator of indicator function

Assume that

$$\psi(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ +\infty & \text{otherwise.} \end{cases}$$

Then, we obtain the **Euclidean projection**

$$\mathsf{Prox}_{\psi}[y] = \arg\min_{x \in \mathcal{C}} \|y - x\|^2.$$

The proximal gradient descent method becomes the projected gradient method:

$$x_t \leftarrow \mathsf{Proj}_{\mathcal{C}}\left[x_{t-1} - \frac{1}{L}\nabla f_0(x_{t-1})\right].$$

Basics of gradient-based optimization: composite problems

Trick 1 to turn a proof for smooth optimization into a proof for composite optimization

The blue inequality for a smooth function tells us

$$f(x) \geq f^\star + \underbrace{\nabla f(x^\star)^\top (x - x^\star)}_{=0} + \frac{\mu}{2}\|x - x^\star\|^2.$$

also known as the second-order growth property. It turns out the property is also true for non-smooth μ-strongly convex functions:

Lemma

If f is a μ-strongly convex function and $x^\star$ is one of its minimizers, then

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

Basics of gradient-based optimization: composite problems

Trick 1 to turn a proof for smooth optimization into a proof for composite optimization

The blue inequality for a smooth function tells us

$$f(x) \geq f^\star + \underbrace{\nabla f(x^\star)^\top (x - x^\star)}_{=0} + \frac{\mu}{2}\|x - x^\star\|^2.$$

also known as the second-order growth property. It turns out the property is also true for non-smooth μ-strongly convex functions:

Consequence

The blue inequality for smooth functions at $x^\star$ still holds for composite functions.

Basics of gradient-based optimization: composite problems

Trick 2 to turn a proof for smooth optimization into a proof for composite optimization

For convex functions ψ, the proximal operator $p(x) = \arg\min_u \frac{1}{2}\|x-u\|^2 + \psi(u)$ is non-expansive

$$\|p(x) - p(y)\| \leq \|x - y\| \qquad \text{for all } x, y.$$

Basics of gradient-based optimization: composite problems

Trick 2 to turn a proof for smooth optimization into a proof for composite optimization

For convex functions ψ, the proximal operator $p(x) = \arg\min_u \frac{1}{2}\|x-u\|^2 + \psi(u)$ is non-expansive

$$\|p(x) - p(y)\| \leq \|x - y\| \qquad \text{for all } x, y.$$

Proof.
$$\frac{1}{2}\|p(x) - y\|^2 + \psi(p(x)) \geq \frac{1}{2}\|p(y) - y\|^2 + \psi(p(y)) + \frac{1}{2}\|p(x) - p(y)\|^2$$
$$\frac{1}{2}\|p(y) - x\|^2 + \psi(p(y)) \geq \frac{1}{2}\|p(x) - x\|^2 + \psi(p(x)) + \frac{1}{2}\|p(x) - p(y)\|^2$$

Add both inequalities, expand the quadratic terms and simplify

$$\langle p(y) - p(x), y - x \rangle \geq \|p(x) - p(y)\|^2.$$

Use Cauchy-Schwarz and conclude (note that you need $p(x)$ to be finite). $\square$

Basics of gradient-based optimization: composite problems

Trick 2 to turn a proof for smooth optimization into a proof for composite optimization

For convex functions ψ, the proximal operator $p(x) = \arg\min_u \frac{1}{2}\|x-u\|^2 + \psi(u)$ is non-expansive

$$\|p(x) - p(y)\| \leq \|x - y\| \qquad \text{for all } x, y.$$

Consequence

If you know how to control $\|x - y\|$ in the smooth case, you know how to control $\|p(x) - p(y)\|$. It turns out that most iterates and even $x^\star$ can be written as $p(x)$.

Part III: Nesterov's Acceleration

Accelerated gradient descent methods

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

Generalization to the composite setting: FISTA

$$x_t \leftarrow \operatorname*{arg\,min}_{x \in \mathbb{R}^p} \frac{1}{2}\left\| x - \left(y_{t-1} - \frac{1}{L}\nabla f_0(y_{t-1}) \right) \right\|_2^2 + \frac{1}{L}\psi(x);$$

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

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

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

see (**????**)

What do we mean by "acceleration"?

Complexity analysis

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

	$\mu > 0$	$\mu = 0$
ISTA	$O\left(\frac{L}{\mu}\log\left(\frac{1}{\varepsilon}\right)\right)$	$O\left(\frac{L}{\varepsilon}\right)$
FISTA	$O\left(\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	$O\left(\sqrt{\frac{L}{\varepsilon}}\right)$

Remarks

- the rate of FISTA is optimal for a "first-order local black box" (**?**).
- 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...

How does “acceleration” work?

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

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

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

$$f(x_t) \leq \varphi_t^\star \overset{\triangle}{=} \min_{x \in \mathbb{R}^p} \varphi_t(x),$$

then

$$f(x_t) - f^\star \leq \lambda_t(\varphi_0(x^\star) - f^\star),$$

where $x^\star$ 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. $f(x_t) \leq \varphi_t^\star \triangleq \min_{x \in \mathbb{R}^p} \varphi_t(x)$.

Remarks

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

How to build an estimate sequence?

Define φ_t recursively

$$\varphi_t(x) \triangleq (1-\alpha_t)\varphi_{t-1}(x) + \alpha_t d_t(x),$$

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 α_t as large as possible, and y_t and x_t such that property 2 holds. Subsequently, $\lambda_t = \prod_{t=1}^t (1-\alpha_t)$.

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. $f(x_t) \leq \varphi_t^\star \triangleq \min_{x\in\mathbb{R}^p} \varphi_t(x)$.

Example: if $\alpha_t = \frac{2}{k+2}$, then $\lambda_t = \prod_{t=1}^t (1-\alpha_t) = \frac{2}{(t+1)(t+2)} = O(1/t^2)$.

- Proofs based on estimates sequences are typically **constructive** and build the algorithm at the same time as they prove convergence, while **describing** the underlying model φ_t.
- But they lead to tedious calculations (about 2 pages).

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. $f(x_t) \leq \varphi_t^\star \triangleq \min_{x\in\mathbb{R}^p} \varphi_t(x)$.

Example: if $\alpha_t = \frac{2}{k+2}$, then $\lambda_t = \prod_{t=1}^{t}(1-\alpha_t) = \frac{2}{(t+1)(t+2)} = O(1/t^2)$.

- Proofs based on estimates sequences are typically **constructive** and build the algorithm at the same time as they prove convergence, while **describing** the underlying model φ_t.
- But they lead to tedious calculations (about 2 pages).

The ODE point of view?

Gradient descent can be interpreted as Euler's method to integrate the gradient flow

$$\dot{x}(t) = -\nabla f(x(t)), \qquad x(0) = x_0.$$

Nesterov's accelerated gradient method admits the following interpretations

- a faster **multistep integration scheme** (**?**).
- or by using a **second-order ODE** (**?**):

$$\ddot{x}(t) + \frac{3}{t}\dot{x}(t) + \nabla f(x(t)) = 0, \qquad x(0) = x_0.$$

The ODE point of view?

Gradient descent can be interpreted as Euler's method to integrate the gradient flow

$$\dot{x}(t) = -\nabla f(x(t)), \qquad x(0) = x_0.$$

Nesterov's accelerated gradient method admits the following interpretations

- a faster **multistep integration scheme** (**?**).
- or by using a **second-order ODE** (**?**):

$$\ddot{x}(t) + \frac{3}{t}\dot{x}(t) + \nabla f(x(t)) = 0, \qquad x(0) = x_0.$$

Unfortunately, this is another point of view (which is already good), but not an explanation.

(**???**)...

Part IV: Stochastic optimization without variance reduction

Stochastic optimization

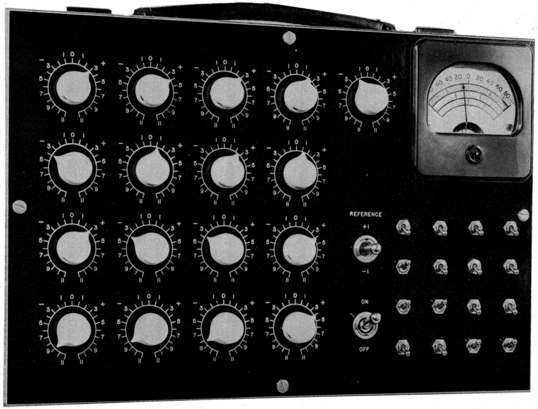


Figure: Adaline, (**?**): A physical device that performs **least square regression using stochastic gradient descent**.

Problems considered in this part

Minimization of expectations with infinite data

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

Problems considered in this part

Minimization of expectations with infinite data

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

In the next part, we will consider

Minimization of (large) finite sums

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

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 η_t, γ_t, classical convergence rates may be obtained:

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

(**??**)...

The stochastic gradient descent algorithm

Comparison of complexity between accelerated gradient descent and stochastic gradient descent for μ-strongly convex objectives, when minimizing a sum of n functions:

FISTA	SGD
$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	$O\left(\frac{\sigma^2}{\mu\varepsilon}\right)$

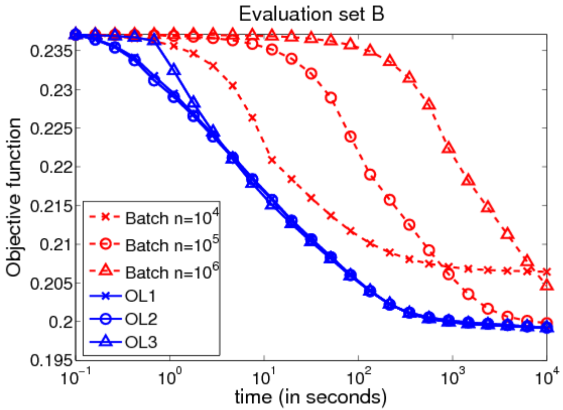
- σ^2 is the variance of the gradient estimators used by SGD, assumed to be bounded here.
- $O(\sigma^2/\mu\varepsilon)$ is the **optimal complexity** for minimizing an expectation (**?**), *e.g.*, with infinite data. FISTA minimizes only the finite sum.

(Realistic) case study

Assuming the (statistical) problem is solved in 100 epochs by SGD with $\mu \approx 1/n$ and $L = 1$; $\Rightarrow \varepsilon = \sigma^2/\mu(100n)$. Then, the complexity of SGD is $100n$, whereas the complexity of FISTA is $\tilde{O}(n^{3/2})$!

The stochastic gradient descent algorithm

Example from **?** about batch vs stochastic optimization:



The plots display the **test objective**. See also Léon Bottou's tutorial from 2007.

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 slides)
- 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** (**?**).

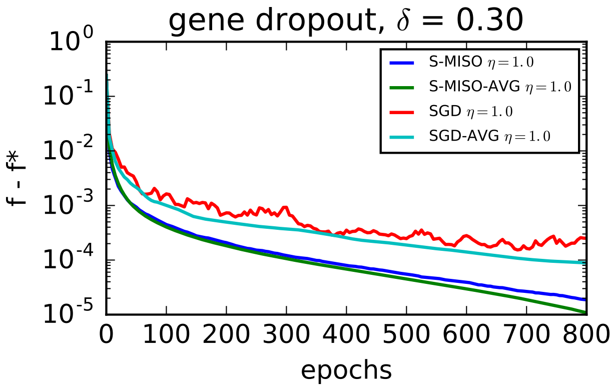
What practice "seems" to tell us

- 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 slides)
- averaging? or not?
- solutions tend to have small norm: implicit regularization?

Practice changes every year. Beware of big inductive claims.

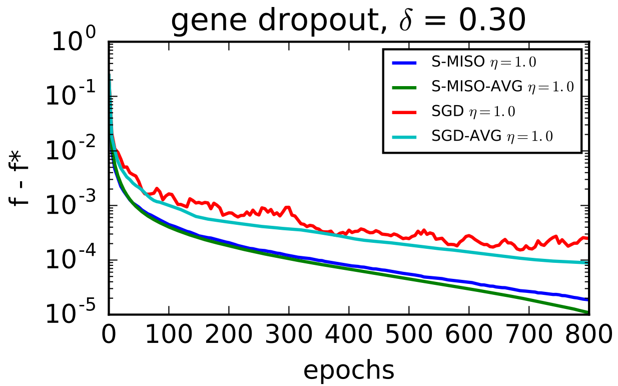
The stochastic gradient descent algorithm

Example of averaging effect



The stochastic gradient descent algorithm

Example of averaging effect



- but if you start averaging too early, convergence may slow down...
- and averaging may break the sparsity for composite problems!

Theoretical reasons for averaging

Obtaining $O(\sigma^2/\mu^2\varepsilon)$ is easy to obtain without averaging. Averaging helps getting rid of the sub-optimal $1/\mu$ factor. How come?

Theoretical reasons for averaging

Obtaining $O(\sigma^2/\mu^2\varepsilon)$ is easy to obtain without averaging. Averaging helps getting rid of the sub-optimal $1/\mu$ factor. How come?

Lemma

Assume that an algorithm generates a sequence $(x_t)_{t\geq 0}$ for minimizing a convex function f, and that there exist sequences $(T_t)_{t\geq 0}$, $(\delta_t)_{t\geq 1}$ in $(0,1)$, $(\beta_t)_{t\geq 1}$ such that.

$$\delta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\delta_t)T_{t-1} + \beta_t, \quad \forall\, t \geq 1.$$

Then, with **no averaging:** $T_t \leq \Gamma_t T_0 + \sum_{k=1}^t \beta_k \Gamma_{t-k}$ *with* $\Gamma_t \triangleq \prod_{k=1}^t (1-\delta_k)$, *and*

$$\mathbb{E}[f(x_t) - f^\star] + \frac{T_t}{\delta_t} \leq \frac{\Gamma_t T_0}{\delta_t} + \sum_{k=1}^t \frac{\beta_k \Gamma_{t-k}}{\delta_t}.$$

see **?**, inspired by **?**.

Theoretical reasons for averaging

Obtaining $O(\sigma^2/\mu^2\varepsilon)$ is easy to obtain without averaging. Averaging helps getting rid of the sub-optimal $1/\mu$ factor. How come?

Lemma

Assume that an algorithm generates a sequence $(x_t)_{t\geq 0}$ for minimizing a convex function f, and that there exist sequences $(T_t)_{t\geq 0}$, $(\delta_t)_{t\geq 1}$ in $(0,1)$, $(\beta_t)_{t\geq 1}$ such that.

$$\delta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\delta_t)T_{t-1} + \beta_t, \quad \forall\, t \geq 1.$$

Then, with **averaging***: introduce $\hat{x}_t = (1-\delta_t)\hat{x}_{t-1} + \delta_t x_t$, and*

$$\mathbb{E}[f(\hat{x}_t) - f^\star] + T_t \leq \Gamma_t(T_0 + f(x_0) - f^\star) + \sum_{t=1}^{k} \beta_t \Gamma_{t-k}.$$

see **?**, inspired by **?**.

Proof of the averaging lemma

Divide by $\Gamma_t = \prod_{k=1}^t (1-\delta_k)$,

$$\frac{\delta_t}{\Gamma_t}\mathbb{E}[f(x_t) - f^\star] + \frac{T_t}{\Gamma_t} \leq \frac{T_{t-1}}{\Gamma_{t-1}} + \frac{\beta_t}{\Gamma_t}.$$

Proof of the averaging lemma

Divide by $\Gamma_t = \prod_{k=1}^t (1-\delta_k)$,

$$\frac{\delta_t}{\Gamma_t}\mathbb{E}[f(x_t) - f^\star] + \frac{T_t}{\Gamma_t} \leq \frac{T_{t-1}}{\Gamma_{t-1}} + \frac{\beta_t}{\Gamma_t}.$$

Sum from $t=1$ to k and notice that we have a **telescopic sum**:

$$\sum_{k=1}^t \frac{\delta_k}{\Gamma_k}\mathbb{E}[f(x_k) - f^\star] + \frac{T_t}{\Gamma_t} \leq T_0 + \sum_{k=1}^t \frac{\beta_k}{\Gamma_k}.$$

Proof of the averaging lemma

Divide by $\Gamma_t = \prod_{k=1}^t (1-\delta_k)$,

$$\frac{\delta_t}{\Gamma_t}\mathbb{E}[f(x_t) - f^\star] + \frac{T_t}{\Gamma_t} \leq \frac{T_{t-1}}{\Gamma_{t-1}} + \frac{\beta_t}{\Gamma_t}.$$

Sum from $t=1$ to k and notice that we have a **telescopic sum**:

$$\sum_{k=1}^t \frac{\delta_k}{\Gamma_k}\mathbb{E}[f(x_k) - f^\star] + \frac{T_t}{\Gamma_t} \leq T_0 + \sum_{k=1}^t \frac{\beta_k}{\Gamma_k}.$$

Then, add $f(x_0) - f^\star$ on both sides and multiply by Γ_t:

$$\sum_{k=1}^t \delta_k \Gamma_{t-k}\mathbb{E}[f(x_k) - f^\star] + \Gamma_t(f(x_0) - f^\star) + T_t \leq \Gamma_t\left(T_0 + f(x_0) - f^\star\right) + \sum_{k=1}^t \beta_k \Gamma_{t-k}.$$

Proof of the averaging lemma

Divide by $\Gamma_t = \prod_{k=1}^t (1-\delta_k)$,

$$\frac{\delta_t}{\Gamma_t}\mathbb{E}[f(x_t) - f^\star] + \frac{T_t}{\Gamma_t} \leq \frac{T_{t-1}}{\Gamma_{t-1}} + \frac{\beta_t}{\Gamma_t}.$$

Sum from $t=1$ to k and notice that we have a **telescopic sum**:

$$\sum_{k=1}^t \frac{\delta_k}{\Gamma_k}\mathbb{E}[f(x_k) - f^\star] + \frac{T_t}{\Gamma_t} \leq T_0 + \sum_{k=1}^t \frac{\beta_k}{\Gamma_k}.$$

Then, add $f(x_0) - f^\star$ on both sides and multiply by Γ_t:

$$\sum_{k=1}^t \delta_k \Gamma_{t-k}\mathbb{E}[f(x_k) - f^\star] + \Gamma_t(f(x_0) - f^\star) + T_t \leq \Gamma_t\left(T_0 + f(x_0) - f^\star\right) + \sum_{k=1}^t \beta_k \Gamma_{t-k}.$$

Note that $\sum_{k=1}^t \delta_k \Gamma_{t-k} + \Gamma_t = 1$ and use **Jensen's inequality**:

$$\mathbb{E}[f(\hat{x}_t) - f^\star] + T_t \leq \Gamma_t\left(T_0 + f(x_0) - f^\star\right) + \sum_{k=1}^t \beta_k \Gamma_{t-k}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1 - \mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, $\eta_t \leq 1/L$ is the step-size, and σ^2 is the noise variance.

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t\mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, $\eta_t \leq 1/L$ is the step-size, and σ^2 is the noise variance.

With constant step-size $\eta_t = 1/L$ (hence, $\delta_t = \mu/L$)

- **With no averaging**:

$$\mathbb{E}[f(x_t) - f^\star] + \frac{L}{2}\mathbb{E}[\|x_t - x^\star\|^2] \leq \left(1 - \frac{\mu}{L}\right)^t \frac{L\|x_0 - x^\star\|^2}{2} + \frac{L}{\mu}\frac{\mu\sigma^2}{L^2}\sum_{k=1}^{t}\left(1-\frac{\mu}{L}\right)^{t-k}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, $\eta_t \leq 1/L$ is the step-size, and σ^2 is the noise variance.

With constant step-size $\eta_t = 1/L$ (hence, $\delta_t = \mu/L$)

- **With no averaging**:

$$\mathbb{E}[f(x_t) - f^\star] + \frac{L}{2}\mathbb{E}[\|x_t - x^\star\|^2] \leq \left(1 - \frac{\mu}{L}\right)^t \frac{L\|x_0 - x^\star\|^2}{2} + \frac{L}{\mu}\frac{\sigma^2}{L}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, $\eta_t \leq 1/L$ is the step-size, and σ^2 is the noise variance.

With constant step-size $\eta_t = 1/L$ (hence, $\delta_t = \mu/L$)

- **With no averaging**:

$$\mathbb{E}[f(x_t) - f^\star] + \frac{L}{2}\mathbb{E}[\|x_t - x^\star\|^2] \leq \left(1 - \frac{\mu}{L}\right)^t \frac{L\|x_0 - x^\star\|^2}{2} + \frac{L}{\mu}\frac{\sigma^2}{L}.$$

- **With averaging**:

$$\mathbb{E}[f(\hat{x}_t) - f^\star] + \frac{\mu}{2}\mathbb{E}[\|x_t - x^\star\|^2] \leq \left(1 - \frac{\mu}{L}\right)^t \left(f(x_0) - f^\star + \frac{\mu}{2}\|x_0 - x^\star\|^2\right) + \frac{\sigma^2}{L}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, η_t is the step-size, and σ^2 is the noise variance. (proof is a few lines).

With finite horizon $T \geq O(L/\mu)$: $\eta = \frac{2}{\mu(2+T)}$

Note that $\delta_t = \frac{2}{(2+T)}$ and that $\Gamma_T = \frac{2}{(T+1)(T+2)} = \frac{\delta_T}{(T+1)} \leq \frac{2}{(T+1)^2}$.

- **With no averaging**:

$$\mathbb{E}[f(x_T) - f^\star] + \frac{\mu}{2\delta_T}\mathbb{E}[\|x_T - x^\star\|^2] \leq \frac{\mu\|x_0 - x^\star\|^2}{2(T+1)} + \frac{1}{\delta_T}\frac{\sigma^2}{\mu(T+1)^2}\sum_{k=1}^{T}(1-\eta)^{T-k}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, η_t is the step-size, and σ^2 is the noise variance. (proof is a few lines).

With finite horizon $T \geq O(L/\mu)$: $\eta = \frac{2}{\mu(2+T)}$

Note that $\delta_t = \frac{2}{(2+T)}$ and that $\Gamma_T = \frac{2}{(T+1)(T+2)} = \frac{\delta_T}{(T+1)} \leq \frac{2}{(T+1)^2}$.

- **With no averaging**:

$$\mathbb{E}[f(x_T) - f^\star] + \frac{\mu}{2\delta_T}\mathbb{E}[\|x_T - x^\star\|^2] \leq \frac{\mu\|x_0 - x^\star\|^2}{2(T+1)} + \frac{1}{\delta_T}\frac{\sigma^2}{\mu(T+1)}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1-\mu\eta_t)T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\, t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, η_t is the step-size, and σ^2 is the noise variance. (proof is a few lines).

With finite horizon $T \geq O(L/\mu)$: $\eta = \frac{2}{\mu(2+T)}$

Note that $\delta_t = \frac{2}{(2+T)}$ and that $\Gamma_T = \frac{2}{(T+1)(T+2)} = \frac{\delta_T}{(T+1)} \leq \frac{2}{(T+1)^2}$.

- **With no averaging**:

$$\mathbb{E}[f(x_T) - f^\star] + \frac{\mu}{2\delta_T}\mathbb{E}[\|x_T - x^\star\|^2] \leq \frac{\mu\|x_0 - x^\star\|^2}{2(T+1)} + \frac{1}{\delta_T}\frac{\sigma^2}{\mu(T+1)}.$$

- **With averaging**:

$$\mathbb{E}[f(\hat{x}_T) - f^\star] + \frac{\mu}{2}\mathbb{E}[\|x_T - x^\star\|^2] \leq \frac{\mu^2\|x_0 - x^\star\|^2}{(T+1)^2} + \frac{\sigma^2}{\mu(T+1)}.$$

Theoretical reasons for averaging: back to SGD

It is possible to show that for SGD (and its proximal variant to come in a few slides), we have

$$\mu\eta_t \mathbb{E}[f(x_t) - f^\star] + T_t \leq (1 - \mu\eta_t) T_{t-1} + \mu\eta_t^2\sigma^2, \quad \forall\ t \geq 1.$$

for $T_k = \frac{\mu}{2}\|x_k - x^\star\|^2$, η_t is the step-size, and σ^2 is the noise variance. (proof is a few lines).

It is possible to obtain converging algorithms with decreasing step sizes, as will be shown next, leading to the complexity

$$O\left(\frac{L}{\mu}\log\left(\frac{f(x_0) - f^\star}{\varepsilon}\right)\right) + O\left(\frac{\sigma^2}{\mu\varepsilon}\right).$$

The stochastic gradient descent algorithm for composite problems

There are many **variants for composite problems** (**??**, *e.g.*), for minimizing

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

where f is L-smooth and μ-strongly convex, and ψ is convex. Consider then the algorithm

$$\boxed{x_t \leftarrow \text{Prox}_{\eta_t \psi}\left[x_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(x_{t-1}),}$$

The stochastic gradient descent algorithm for composite problems

There are many **variants for composite problems** (**??**, *e.g.*), for minimizing

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

where f is L-smooth and μ-strongly convex, and ψ is convex. Consider then the algorithm

$$\boxed{x_t \leftarrow \text{Prox}_{\eta_t \psi}\left[x_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(x_{t-1}),}$$

With $\eta_t = 1/L$ and the averaging strategy $\tilde{x}_t = (1-\mu/L)\tilde{x}_{t-1} + (\mu/L)x_t$,

$$\mathbb{E}\left[f(\tilde{x}_t) - f^\star + \frac{\mu}{2}\|x_t - x^\star\|^2\right] \leq 2\left(1 - \frac{\mu}{L}\right)^t (f(x_0) - f^\star) + \frac{\sigma^2}{L},$$

assuming σ to be bounded, see for instance (**?**).

The stochastic gradient descent algorithm for composite problems

With constant step size, the algorithm converges to a noise-dominated region, as fast as if the problem was deterministic.

The stochastic gradient descent algorithm for composite problems

With constant step size, the algorithm converges to a noise-dominated region, as fast as if the problem was deterministic.

Then, it oscillates, which requires to **reduce the variance** of the updates. This can be done by reducing the step sizes:

Lemma

Use a constant step-size strategy until $\mathbb{E}[f(\tilde{x}_t) - f^\star] \leq 2\sigma^2/L$*; then restart and use the decreasing step-sizes* $\eta_t = \min\left(\frac{1}{L}, \frac{2}{\mu(t+2)}\right)$*. The total number of iterations to find a point* $\hat{x}$ *such that* $\mathbb{E}[f(\hat{x}) - f^\star] \leq \varepsilon$ *is upper-bounded by*

$$O\left(\frac{L}{\mu}\log\left(\frac{f(x_0) - f^\star}{\varepsilon}\right)\right) + O\left(\frac{\sigma^2}{\mu\varepsilon}\right).$$

see for instance (**?**).

Other variants of the stochastic gradient descent algorithm

Inspired by Jamie Soel's presentation at NIPS'2018

- SGD:

$$x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}).$$

Other variants of 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:

$$x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}).$$

Other variants of 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:

$$x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1} + \beta_t(x_{t-1} - x_{t-2})) + \beta_t(x_{t-1} - x_{t-2}).$$

Other variants of 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 (**?**)

$$x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}).$$

Other variants of 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 (**?**)
$$x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}).$$
- Adam (**?**):
$$x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}) + \beta_t H_t^{-1} H_{t-1}(x_{t-1} - x_{t-2}).$$

Other variants of 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 (**?**)

$$x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}).$$

- Adam (**?**):

$$x_t = x_{t-1} - \eta_t H_t^{-1} \nabla f_t(x_{t-1}) + \beta_t H_t^{-1} H_{t-1}(x_{t-1} - x_{t-2}).$$

The proximal accelerated stochastic gradient descent algorithm

$O(\sigma^2/\mu\varepsilon)$ is already optimal...

The proximal accelerated stochastic gradient descent algorithm

$O(\sigma^2/\mu\varepsilon)$ is already optimal...

Can we forget faster the initial condition?

Going from

$$O\left(\frac{L}{\mu}\log\left(\frac{f(x_0)-f^\star}{\varepsilon}\right)\right)+O\left(\frac{\sigma^2}{\mu\varepsilon}\right).$$

to

$$O\left(\sqrt{\frac{L}{\mu}}\log\left(\frac{f(x_0)-f^\star}{\varepsilon}\right)\right)+O\left(\frac{\sigma^2}{\mu\varepsilon}\right).$$

The first algorithm achieving this complexity was proposed by **?**.

The proximal accelerated stochastic gradient descent algorithm

Here is another one (**?**):

$$\boxed{\begin{array}{l} x_t = \text{Prox}_{\eta_t \psi}\left[y_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(y_{t-1}) \\ y_t = x_t + \beta_t(x_t - x_{t-1}) \quad \text{with} \quad \beta_t = \dfrac{(1-\sqrt{\mu\eta_t})\sqrt{\eta_{t+1}}}{(1+\sqrt{\mu\eta_{t+1}})\sqrt{\eta_t}}. \end{array}}$$

It achieves the previous optimal complexity with (i) one restart, (ii) decreasing step-sizes $\eta_t = \min\left(\frac{1}{L}, \frac{2}{\mu(t+2)^2}\right)$, and (iii) **without averaging**.

The proximal accelerated stochastic gradient descent algorithm

Here is another one (**?**):

$$\boxed{\begin{aligned} x_t &= \text{Prox}_{\eta_t \psi}\left[y_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(y_{t-1}) \\ y_t &= x_t + \beta_t (x_t - x_{t-1}) \quad \text{with} \quad \beta_t = \frac{(1-\sqrt{\mu\eta_t})\sqrt{\eta_{t+1}}}{(1+\sqrt{\mu\eta_{t+1}})\sqrt{\eta_t}}. \end{aligned}}$$

It achieves the previous optimal complexity with (i) one restart, (ii) decreasing step-sizes $\eta_t = \min\left(\frac{1}{L}, \frac{2}{\mu(t+2)^2}\right)$, and (iii) **without averaging**.

Does it work?

- not always.

The proximal accelerated stochastic gradient descent algorithm

Here is another one (**?**):

$$\boxed{\begin{aligned} x_t &= \text{Prox}_{\eta_t \psi}\left[y_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(y_{t-1}) \\ y_t &= x_t + \beta_t(x_t - x_{t-1}) \quad \text{with} \quad \beta_t = \frac{(1-\sqrt{\mu\eta_t})\sqrt{\eta_{t+1}}}{(1+\sqrt{\mu\eta_{t+1}})\sqrt{\eta_t}}. \end{aligned}}$$

It achieves the previous optimal complexity with (i) one restart, (ii) decreasing step-sizes $\eta_t = \min\left(\frac{1}{L}, \frac{2}{\mu(t+2)^2}\right)$, and (iii) **without averaging**.

why?

- we lied to you about the safety of the bounded noise variance assumption.
- the accelerated algorithm with constant step size (which is used to forget the initial condition) has much worth dependency in σ^2 (see next slide).

The proximal accelerated stochastic gradient descent algorithm

Here is another one (**?**):

$$\boxed{\begin{aligned} x_t &= \text{Prox}_{\eta_t \psi}\left[y_{t-1} - \eta_t g_t\right] \quad \text{with} \quad \mathbb{E}[g_t | \mathcal{F}_{t-1}] = \nabla f_0(y_{t-1}) \\ y_t &= x_t + \beta_t(x_t - x_{t-1}) \quad \text{with} \quad \beta_t = \frac{(1-\sqrt{\mu\eta_t})\sqrt{\eta_{t+1}}}{(1+\sqrt{\mu\eta_{t+1}})\sqrt{\eta_t}}. \end{aligned}}$$

It achieves the previous optimal complexity with (i) one restart, (ii) decreasing step-sizes $\eta_t = \min\left(\frac{1}{L}, \frac{2}{\mu(t+2)^2}\right)$, and (iii) **without averaging**.

Is it worthless?

- **removing the need for averaging** is great for sparse problems.
- with a **mini-batch** of size $\sqrt{L/\mu}$, we obtain the same complexity as the unaccelerated algorithm and the same stability w.r.t. σ^2, and we can parallelize for free!

The bounded noise assumption

Consider a quadratic function

$$\min_{x \in \mathbb{R}^p} \left\{ f(x) \triangleq \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (a_i^\top x)^2 \right\}.$$

Exact and stochastic gradients (drawn by randomply selecting one index i) are respectively

$$\nabla f(x) = \frac{1}{n} \mathbf{A}^\top \mathbf{A} x \qquad g = a_i a_i^\top x.$$

The amplitude of the gradient error $g - \nabla f(x)$ is proportional to x, and thus unbounded.

What can we do?

- study precisely quadratic problems (**?**).
- make weaker assumptions (**?**).
- hope that during optimization, the trajectory remains with bounded σ^2.

The problem with accelerated stochastic algorithms

Convergence of proximal SGD with $\eta_t = 1/L$

$$\mathbb{E}[f(\hat{x}_t) - f^\star] \leq 2\left(1 - \frac{\mu}{L}\right)^t (f(x_0) - f^\star) + \frac{\sigma^2}{L}.$$

Convergence of accelerated proximal SGD with $\eta_t = 1/L$

$$\mathbb{E}[f(\hat{x}_t) - f^\star] \leq 2\left(1 - \sqrt{\frac{\mu}{L}}\right)^t (f(x_0) - f^\star) + \frac{\sigma^2}{\sqrt{\mu L}}.$$

The problem with accelerated stochastic algorithms

Convergence of proximal SGD with $\eta_t = 1/L$

$$\mathbb{E}[f(\hat{x}_t) - f^\star] \leq 2\left(1 - \frac{\mu}{L}\right)^t (f(x_0) - f^\star) + \frac{\sigma^2}{L}.$$

Convergence of accelerated proximal SGD with $\eta_t = 1/L$

$$\mathbb{E}[f(\hat{x}_t) - f^\star] \leq 2\left(1 - \sqrt{\frac{\mu}{L}}\right)^t (f(x_0) - f^\star) + \frac{\sigma^2}{\sqrt{\mu L}}.$$

Effect of mini-batches of size $\sqrt{L/\mu}$ for accelerated proximal SGD

- same stability as unaccelerated SGD with respect to σ^2;
- cost per iteration $\times\sqrt{L/\mu}$ leads to same complexity as unaccelerated SGD;
- **easy to parallelize**.
- **in practice seems better than both approaches**.

Part V: Stochastic optimization with variance reduction

Back to finite sums

Consider now that the training set is finite:

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

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 $n = 1$

The rates are optimal for a “first-order local black box” (**?**).

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 ∇f_i computed per iteration, with **fast convergence rates**, e.g., SAG (**?**):

$$x_t \leftarrow x_{t-1} - \frac{\gamma}{Ln} \sum_{i=1}^{n} y_i^t \quad \text{with} \quad y_i^t = \left\{ \begin{array}{ll} \nabla f_i(x_{t-1}) & \text{if } \; i = i_t \\ y_i^{t-1} & \text{otherwise} \end{array} \right. .$$

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 ∇f_i computed per iteration, with **fast convergence rates**, e.g., SAG (**?**):

$$x_t \leftarrow x_{t-1} - \frac{\gamma}{Ln} \sum_{i=1}^{n} y_i^t \quad \text{with} \quad y_i^t = \begin{cases} \nabla f_i(x_{t-1}) & \text{if } \; i = i_t \\ y_i^{t-1} & \text{otherwise} \end{cases}.$$

See also SVRG, SAGA, SDCA, MISO, Finito...
Some of these algorithms perform updates of the form

$$x_t \leftarrow x_{t-1} - \eta_t g_t \quad \text{with} \quad \mathbb{E}[g_t] = \nabla f(x_{t-1}),$$

but g_t has **lower variance** than in SGD.
(**???????**)

Incremental gradient descent methods

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

	$\mu > 0$
FISTA	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$
SVRG, SAG, SAGA, SDCA, MISO, Finito	$O\left(\max\left(n, \frac{\bar{L}}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

Incremental gradient descent methods

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

	$\mu > 0$
FISTA	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$
SVRG, SAG, SAGA, SDCA, MISO, Finito	$O\left(\max\left(n, \frac{\bar{L}}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

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** ψ.
- 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 $\mathbb{E}[f(x_t) - f^\star] \leq \varepsilon$ is

	$\mu > 0$
FISTA	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$
SVRG, SAG, SAGA, SDCA, MISO, Finito	$O\left(\max\left(n, \frac{\bar{L}}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

Important remarks

- When $f_i(x) = \ell(z_i^\top x)$, the memory footprint is $O(n)$ otherwise $O(dn)$, except for SVRG $O(d)$.
- Most algorithms can become adaptive to unknown μ (**?**).
- $\bar{L}$ is the average (or max) of the Lipschitz constants of the ∇f_i's.
- The L for FISTA is the Lipschitz constant of ∇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

	$\mu > 0$
FISTA	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$
SVRG, SAG, SAGA, SDCA, MISO, Finito	$O\left(\max\left(n, \frac{\bar{L}}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$
Accelerated versions	$O\left(\max\left(n, \sqrt{n\frac{\bar{L}}{\mu}}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

- Acceleration for specific algorithms (**????**).
- Generic acceleration: Catalyst (**?**) with $\tilde{O}$.
- see (**?**) 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

	$\mu > 0$
FISTA	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$
SVRG, SAG, SAGA, SDCA, MISO, Finito	$O\left(\max\left(n, \frac{\bar{L}}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$
Accelerated versions	$O\left(\max\left(n, \sqrt{n\frac{\bar{L}}{\mu}}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

- 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 since constant step size.
- for non-convex objectives: nothing clear yet.

When is acceleration useful?

- when the problem is badly conditioned (L/μ large).
- when the amount of data is large, but not too large (such that one-pass un-regularized SGD does not work).

The stochastic finite-sum problem

Assume we want to tackle

$$\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \psi(x) \quad \text{with} \quad f_i(x) = \mathbb{E}_\rho[\tilde{f}_i(x, \rho)],$$

such that the previous algorithms do not apply anymore. Each f_i corresponds ot a data point but each sample is **corrupted by a random perturbation** ρ.

The stochastic finite-sum problem

Assume we want to tackle

$$\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \psi(x) \quad \text{with} \quad f_i(x) = \mathbb{E}_\rho[\tilde{f}_i(x, \rho)],$$

such that the previous algorithms do not apply anymore. Each f_i corresponds ot a data point but each sample is **corrupted by a random perturbation** ρ.
Assume that we can access unbiased estimates of the gradients $f_i(x)$ with variance $\tilde{\sigma}^2$ **much smaller than the noise due to data sampling.**

The stochastic finite-sum problem

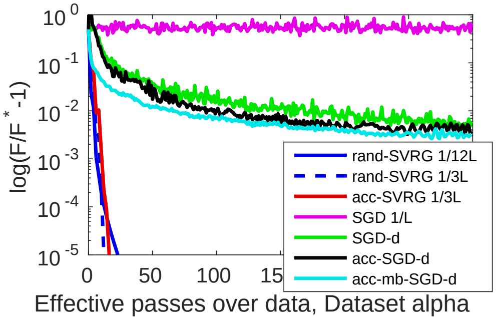
Assume we want to tackle

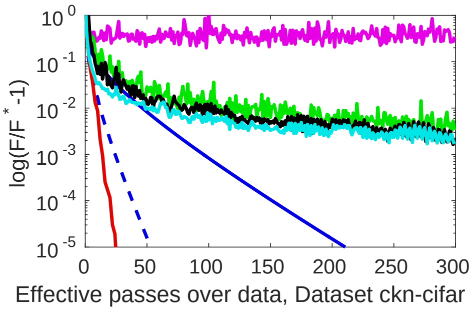
$$\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \psi(x) \quad \text{with} \quad f_i(x) = \mathbb{E}_\rho[\tilde{f}_i(x, \rho)],$$

such that the previous algorithms do not apply anymore. Each f_i corresponds ot a data point but each sample is **corrupted by a random perturbation** ρ.
Assume that we can access unbiased estimates of the gradients $f_i(x)$ with variance $\tilde{\sigma}^2$ **much smaller than the noise due to data sampling.**
Then, it is possible to adapt the previous algorithms to this setting; the optimal complexity becomes:

$$O\left(\left(n + \sqrt{n\frac{L}{\mu}}\right) \log\left(\frac{F(x_0) - F^\star}{\varepsilon}\right)\right) + O\left(\frac{\tilde{\sigma}^2}{\mu\varepsilon}\right),$$

A few experiments

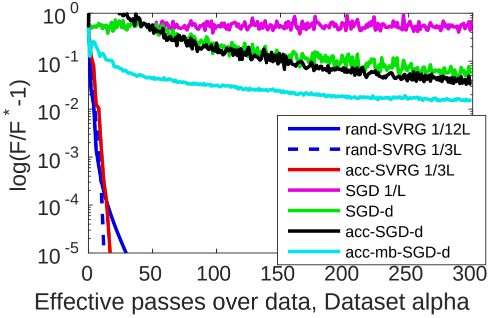


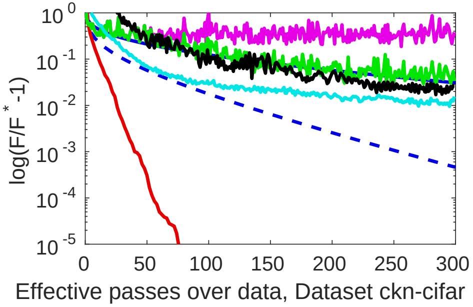


ℓ_2-logistic regression on two datasets, with $\mu = 1/10n$.

- no big difference between the variants of SGD with decreasing step sizes;
- variance reduction makes a huge difference.
- acceleration helps on ckn-cifar.

A few experiments

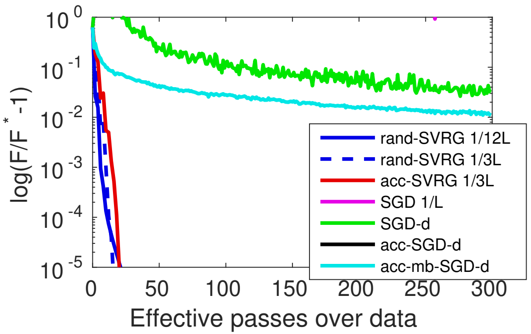


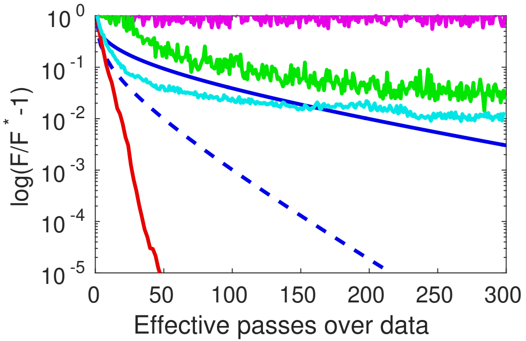


ℓ_2-logistic regression on two datasets, with $\mu = 1/100n$.

- as conditioning worsens, the benefits of acceleration are larger.
- accelerated SGD with mini-batches take the lead among SGD methods.

A few experiments





SVM with squared hinge loss on two datasets, with $\mu = 1/10n$.

- here, gradients are potentially unbounded and accelerated SGD diverges!
- accelerated SGD with mini-batches is stable and faster than SGD.

Part VI: Catalyst and QNing

- H. Lin, J. Mairal, and Z. Harchaoui. Catalyst Acceleration for First-order Convex Optimization: from Theory to Practice. JMLR. 2018.
- H. Lin, J. Mairal, and Z. Harchaoui. An Inexact Variable Metric Proximal Point Algorithm for Generic Quasi-Newton Acceleration. SIAM Journal on Optimization. 2019.

Part VI: Catalyst and QNing

- H. Lin, J. Mairal, and Z. Harchaoui. Catalyst Acceleration for First-order Convex Optimization: from Theory to Practice. JMLR. 2018.
- H. Lin, J. Mairal, and Z. Harchaoui. An Inexact Variable Metric Proximal Point Algorithm for Generic Quasi-Newton Acceleration. SIAM Journal on Optimization. 2019.

(we will talk about smoothing techniques and Quasi-Newton)

An old idea

Old idea: Smooth the function and then optimize.

- The strategy appears in early work about variable metric bundle methods. (**?????**) ...

An old idea

Old idea: Smooth the function and then optimize.

- The strategy appears in early work about variable metric bundle methods. (**?????**) ...

The Moreau-Yosida envelope

Given $f : \mathbb{R}^d \to \mathbb{R}$ a convex function, the Moreau-Yosida envelope of f is the function $F : \mathbb{R}^d \to \mathbb{R}$ defined as

$$F(x) = \min_{w \in \mathbb{R}^d} \left\{ f(w) + \frac{\kappa}{2} \|w - x\|^2 \right\}.$$

The **proximal operator** $p(x)$ is the unique minimizer of the problem.

The Moreau-Yosida regularization

$$F(x) = \min_{w \in \mathbb{R}^d} \left\{ f(w) + \frac{\kappa}{2}\|w - x\|^2 \right\}.$$

Basic properties (see **?**)

- Minimizing f and F is equivalent in the sense that

$$\min_{x \in \mathbb{R}^d} F(x) = \min_{x \in \mathbb{R}^d} f(x),$$

  and the solution set of the two problems coincide with each other.

- F is continuously differentiable even when f is not and

$$\nabla F(x) = \kappa(x - p(x)).$$

  In addition, ∇F is Lipschitz continuous with parameter $L_F = \kappa$.

- If f is μ-strongly convex then F is also strongly convex with parameter $\mu_F = \frac{\mu\kappa}{\mu+\kappa}$.

The Moreau-Yosida regularization

$$F(x) = \min_{w \in \mathbb{R}^d} \left\{ f(w) + \frac{\kappa}{2} \|w - x\|^2 \right\}.$$

Basic properties (see **?**)

- Minimizing f and F is equivalent in the sense that

$$\min_{x \in \mathbb{R}^d} F(x) = \min_{x \in \mathbb{R}^d} f(x),$$

  and the solution set of the two problems coincide with each other.

- F is continuously differentiable even when f is not and

$$\nabla F(x) = \kappa(x - p(x)).$$

  In addition, ∇F is Lipschitz continuous with parameter $L_F = \kappa$.

F enjoys nice properties: smoothness, (strong) convexity and we can control its condition number $1/q = 1 + \kappa/\mu$.

The proximal point algorithm

A naive approach consists of **minimizing the smoothed objective F instead of f** with a method designed for smooth optimization.

Consider indeed

$$x_{k+1} = x_k - \frac{1}{\kappa}\nabla F(x_k).$$

By rewriting the gradient $\nabla F(x_k)$ as $\kappa(x_k - p(x_k))$, we obtain

$$x_{k+1} = p(x_k) = \arg\min_{w \in \mathbb{R}^p} \left\{ f(w) + \frac{\kappa}{2}\|w - x_k\|^2 \right\}.$$

This is exactly the **proximal point algorithm** (**??**).

The accelerated proximal point algorithm

Consider now

$$x_{k+1} = y_k - \frac{1}{\kappa}\nabla F(y_k) \quad \text{and} \quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_k),$$

where β_{k+1} is a Nesterov-like extrapolation parameter. We may now rewrite the update using the value of ∇F, which gives:

$$x_{k+1} = p(y_k) \quad \text{and} \quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_k)$$

This is the **accelerated proximal point algorithm** of **?**.

The accelerated proximal point algorithm

Consider now

$$x_{k+1} = y_k - \frac{1}{\kappa}\nabla F(y_k) \quad \text{and} \quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_k),$$

where β_{k+1} is a Nesterov-like extrapolation parameter. We may now rewrite the update using the value of ∇F, which gives:

$$x_{k+1} = p(y_k) \quad \text{and} \quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_k)$$

This is the **accelerated proximal point algorithm** of **?**.

Remarks

- F may be **better conditioned** than f when $1 + \kappa/\mu \leq L/\mu$;
- Computing $p(y_k)$ has a cost!

A fresh look at Catalyst (?)

Catalyst is a particular **accelerated proximal point algorithm with inexact gradients** (**?**).

$$x_{k+1} \approx p(y_k) \quad \text{and} \quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_k)$$

The quantity x_{k+1} is obtained by using an optimization method $\mathcal{M}$ for approximately solving:

$$x_{k+1} \approx \operatorname*{arg\,min}_{w \in \mathbb{R}^p} \left\{ f(w) + \frac{\kappa}{2} \|w - y_k\|^2 \right\},$$

Catalyst provides Nesterov's acceleration to $\mathcal{M}$ with...

- **restart strategies** for solving the sub-problems;
- **global complexity analysis** resulting in theoretical acceleration;
- **optimal balancing between outer and inner computations**.

see also (**?????**)

This work

Contributions

- **Generic acceleration scheme**, which applies to algorithms $\mathcal{M}$ that have **linear convergence rates** for strongly convex problems..
- Provides explicit **support to non-strongly convex objectives**.
- Complexity analysis for μ-strongly convex objectives.
- Complexity analysis for non-strongly convex objectives.

Requirements on $\mathcal{M}$

Objective function f

- f is **convex or μ-strongly convex**.

Linear convergence

- Say a sub-problem consists of minimizing h; we want $\mathcal{M}$ to produce a sequence of iterates $(z_t)_{t\geq 0}$ with **linear convergence rate**

$$h(z_t) - h^\star \leq C_{\mathcal{M}}(1 - \tau_{\mathcal{M}})^t (h(z_0) - h^\star),$$

  which may possibly hold only **in expectation** if $\mathcal{M}$ is randomized.
- **No assumption** is made on the behavior of $\mathcal{M}$ for **non-strongly convex problems**.
- Variants may be allowed when linear convergence is stated in terms of dual certificate.

When do we stop the method $\mathcal{M}$?

Three strategies to balance outer and inner computations

(a) use a **pre-defined sequence** $(\varepsilon_k)_{k \geq 0}$ and stop the optimization method $\mathcal{M}$ when the sub-problems $\min h_k$ satisfies

$$h_k(z_t) - h_k^\star \leq \varepsilon_k.$$

(b) use a **pre-defined sequence** $(\delta_k)_{k \geq 0}$ and stop the optimization method $\mathcal{M}$ when the sub-problems $\min h_k$ satisfies

$$h_k(z_t) - h_k^\star \leq \frac{\delta_k}{2} \|z_t - y_k\|^2.$$

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$.

When do we stop the method $\mathcal{M}$?

Three strategies to balance outer and inner computations

(a) use a **pre-defined sequence** $(\varepsilon_k)_{k \geq 0}$ and stop the optimization method $\mathcal{M}$ when the sub-problems $\min h_k$ satisfies

$$h_k(z_t) - h_k^\star \leq \varepsilon_k.$$

(b) use a **pre-defined sequence** $(\delta_k)_{k \geq 0}$ and stop the optimization method $\mathcal{M}$ when the sub-problems $\min h_k$ satisfies

$$h_k(z_t) - h_k^\star \leq \frac{\delta_k}{2}\|z_t - y_k\|^2.$$

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$.

Remark

- (c) implies (a) and requires $T_{\mathcal{M}}$ to be larger than necessary in practice; it leads to the simplest and most effective strategies.

When do we stop the method $\mathcal{M}$?

Three strategies for μ-strongly convex objectives f

(a) use
$$\varepsilon_k = \frac{1}{2}C(1-\rho)^{k+1} \quad \text{with} \quad C \geq f(x_0) - f^* \text{ and } \rho < \sqrt{q}.$$
where q is the inverse of the condition number of F: $q = \frac{\mu}{(\mu+\kappa)}$

(b) use
$$\delta_k = \frac{\sqrt{q}}{2-\sqrt{q}}.$$

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$ for solving each sub-problem with
$$T_{\mathcal{M}} = \frac{1}{\tau_{\mathcal{M}}} \log\left(19 C_{\mathcal{M}} \frac{L+\kappa}{\kappa}\right).$$ (be more aggressive in practice)

When do we stop the method $\mathcal{M}$?

Three strategies for $\mu = 0$

(a) use
$$\varepsilon_k = \frac{f(x_0) - f^\star}{2(k+1)^{4+\gamma}} \quad \text{with} \quad \gamma > 0.$$

(b) use
$$\delta_k = \frac{1}{(k+1)^2}.$$

(c) use a **pre-defined budget** T_k of iterations of the method $\mathcal{M}$ for solving each sub-problem h_k with
$$T_k = O(\log(k)) \text{ (use a constant in practice)}$$

Other implementation details

See the paper for

- Nesterov's extrapolation parameters (fairly standard).
- restart strategies for solving the sub-problems.

Other implementation details

See the paper for

- Nesterov's extrapolation parameters (fairly standard).
- restart strategies for solving the sub-problems.

Spoiler: optimal balance for inner/outer computations

To choose κ, maximize

$$\frac{\tau_{\mathcal{M}}}{\sqrt{\mu+\kappa}}.$$

Remember that $\tau_{\mathcal{M}}$ drives the convergence rate for the sub-problems

$$h(w_t) - h^\star \leq C_{\mathcal{M}}(1-\tau_{\mathcal{M}})^t(h(w_0) - h^\star).$$

For the standard gradient descent method, use $\kappa = L - 2\mu$.

Outer-loop convergence analysis

With strong convexity

Using strategy (a),

$$f(x_k) - f^* \leqslant C(1-\rho)^{k+1}(f(x_0) - f^*) \quad \text{with} \quad \rho < \sqrt{q},$$

and a similar result holds for (b).

Without strong convexity

Using strategy (b),

$$f(x_k) - f^* \leqslant \frac{4\kappa \|x_0 - x^*\|^2}{(k+1)^2}.$$

and a similar result holds for (a).

Inner-loop convergence analysis

Using appropriate restart strategies, the inner-loop stopping criterions are satisfied after T_k iterations, where

$$T_k = \tilde{O}\left(\frac{1}{\tau_{\mathcal{M}}}\right) \quad \text{when} \quad \mu > 0,$$

and

$$T_k = \tilde{O}\left(\frac{\log(k)}{\tau_{\mathcal{M}}}\right) \quad \text{when} \quad \mu = 0.$$

The $\tilde{O}$ hides logarithmic quantities in μ, κ and universal constants.

Global complexity analysis

By combining the two previous strategies, we obtain that the guarantee $f(x_k) - f^\star \leq \varepsilon$ is achieved after N iterations of the method $\mathcal{M}$, where

$$N = \tilde{O}\left(\frac{1}{\tau_{\mathcal{M}}\sqrt{q}} \log\left(\frac{1}{\varepsilon}\right)\right) \quad \text{when} \quad \mu > 0,$$

and

$$N = \tilde{O}\left(\frac{1}{\tau_{\mathcal{M}}}\sqrt{\frac{\kappa}{\varepsilon}} \log\left(\frac{1}{\varepsilon}\right)\right) \quad \text{when} \quad \mu = 0.$$

Similar results hold also for randomized algorithms.

Global complexity analysis

By combining the two previous strategies, we obtain that the guarantee $f(x_k) - f^\star \leq \varepsilon$ is achieved after N iterations of the method $\mathcal{M}$, where

$$N = \tilde{O}\left(\frac{1}{\tau_{\mathcal{M}}\sqrt{q}} \log\left(\frac{1}{\varepsilon}\right)\right) \quad \text{when} \quad \mu > 0,$$

and

$$N = \tilde{O}\left(\frac{1}{\tau_{\mathcal{M}}}\sqrt{\frac{\kappa}{\varepsilon}} \log\left(\frac{1}{\varepsilon}\right)\right) \quad \text{when} \quad \mu = 0.$$

Similar results hold also for randomized algorithms.

Theoretical choice of κ

maximize

$$\frac{\tau_{\mathcal{M}}}{\sqrt{\mu + \kappa}}.$$

For gradient descent, $\tau_{\mathcal{M}} = \frac{\mu+\kappa}{L+\kappa} \Rightarrow \kappa = L - 2\mu \Rightarrow \frac{1}{\tau_{\mathcal{M}}\sqrt{q}} \leq 2\sqrt{\frac{L}{\mu}}$

Applications

Expected computational complexity in the regime $n \leq L/\mu$ when $\mu > 0$,

	$\mu > 0$	$\mu = 0$	Catalyst $\mu > 0$	Cat. $\mu = 0$
FG	$O\left(n\left(\frac{L}{\mu}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$	$O\left(n\frac{L}{\varepsilon}\right)$	$\tilde{O}\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	$\tilde{O}\left(n\sqrt{\frac{L}{\varepsilon}}\right)$
SAG	$O\left(\frac{L}{\mu}\log\left(\frac{1}{\varepsilon}\right)\right)$		$\tilde{O}\left(\sqrt{\frac{nL}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	$\tilde{O}\left(\sqrt{\frac{nL}{\varepsilon}}\right)$
SAGA				
Finito/MISO		NA		
SDCA				
SVRG				
Acc-FG	$O\left(n\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	$O\left(n\sqrt{\frac{L}{\varepsilon}}\right)$	no acceleration	
Acc-SDCA	$\tilde{O}\left(\sqrt{\frac{nL}{\mu}}\log\left(\frac{1}{\varepsilon}\right)\right)$	NA		

QNing

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- **Quasi-Newton** methods work with the parameter and gradient differences between successive iterations:

$$s_k \triangleq x_{k+1} - x_k, \quad y_k \triangleq \nabla f(x_{k+1}) - \nabla f(x_k).$$

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- **Quasi-Newton** methods work with the parameter and gradient differences between successive iterations:

$$s_k \triangleq x_{k+1} - x_k, \quad y_k \triangleq \nabla f(x_{k+1}) - \nabla f(x_k).$$

- They start with an initial approximation $B_0 \triangleq \sigma I$, and choose B_{k+1} to **interpolate the gradient difference**:

$$B_{k+1} s_k = y_k.$$

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- **Quasi-Newton** methods work with the parameter and gradient differences between successive iterations:

$$s_k \triangleq x_{k+1} - x_k, \quad y_k \triangleq \nabla f(x_{k+1}) - \nabla f(x_k).$$

- They start with an initial approximation $B_0 \triangleq \sigma I$, and choose B_{k+1} to **interpolate the gradient difference**:

$$B_{k+1} s_k = y_k.$$

- Since B_{k+1} is not unique, the Broyden-Fletcher-Goldfarb-Shanno **(BFGS)** method chooses the symmetric matrix whose difference with B_k is minimal:

$$B_{k+1} = B_k - \frac{B_k s_k s_k B_k}{s_k B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}.$$

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.
- They perform updates of the form

$$x_{k+1} \leftarrow x_k - \eta_k B_k^{-1} \nabla f(x_k).$$

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.
- They perform updates of the form

$$x_{k+1} \leftarrow x_k - \eta_k B_k^{-1} \nabla f(x_k).$$

- The BFGS method has a **superlinear convergence rate**.

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.
- They perform updates of the form

$$x_{k+1} \leftarrow x_k - \eta_k B_k^{-1} \nabla f(x_k).$$

- The BFGS method has a **superlinear convergence rate**.
- But, it still uses a dense $p \times p$ matrix B_k.

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.
- They perform updates of the form

$$x_{k+1} \leftarrow x_k - \eta_k B_k^{-1} \nabla f(x_k).$$

- The BFGS method has a **superlinear convergence rate**.
- But, it still uses a dense $p \times p$ matrix B_k.
- Instead of storing B_k, the **limited-memory BFGS** (L-BFGS) method stores the previous l differences s_k and y_k.

Quasi-Newton and L-BFGS

Presentation borrowed from Mark Schmidt, NIPS OPT 2010

- Update skipping/damping or a sophisticated line search (Wolfe conditions) can keep B_{k+1} positive-definite.
- They perform updates of the form

$$x_{k+1} \leftarrow x_k - \eta_k B_k^{-1} \nabla f(x_k).$$

- The BFGS method has a **superlinear convergence rate**.
- But, it still uses a dense $p \times p$ matrix B_k.
- Instead of storing B_k, the **limited-memory BFGS** (L-BFGS) method stores the previous l differences s_k and y_k.
- We can solve a linear system involving these updates when B_0 is diagonal in $O(dl)$ (**?**).

Limited-Memory BFGS (L-BFGS)

Remarks

- using the right initialization B_0 is crucial.
- the calibration of the line-search is also an art.

Limited-Memory BFGS (L-BFGS)

Remarks

- using the right initialization B_0 is crucial.
- the calibration of the line-search is also an art.

Pros

- **one of the largest practical success of smooth optimization**.

Limited-Memory BFGS (L-BFGS)

Remarks

- using the right initialization B_0 is crucial.
- the calibration of the line-search is also an art.

Pros

- **one of the largest practical success of smooth optimization**.

Cons

- worst-case convergence rates for strongly-convex functions are linear, but **no better than the gradient descent method**.
- proximal variants typically requires solving many times

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}(x-z)B_k(z-z) + \psi(x).$$

- no guarantee of approximating the Hessian.

QNing

Main recipe

- L-BFGS applied to the **smoothed objective** F with **inexact gradients** (see **?**).
- inexact gradients are obtained by **solving sub-problems** using a first-order optimization method $\mathcal{M}$;
- ideally, $\mathcal{M}$ is **able to adapt to the problem structure** (finite sum, composite regularization).
- replace L-BFGS steps by proximal point steps if no sufficient decrease is estimated $\Rightarrow$ **no line search on** F;

An old idea (again)

Old idea: Smooth the function and then optimize.

- The strategy appears in early work about variable metric bundle methods. (**?????**) ...

The Moreau-Yosida envelope

Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ a convex function, the Moreau-Yosida envelope of f is the function $F : \mathbb{R}^d \rightarrow \mathbb{R}$ defined as

$$F(x) = \min_{w \in \mathbb{R}^d} \left\{ f(w) + \frac{\kappa}{2} \|w - x\|^2 \right\}.$$

The **proximal operator** $p(x)$ is the unique minimizer of the problem.

QNing

Main recipe

- L-BFGS applied to the **smoothed objective** F with **inexact gradients** (see **?**).
- inexact gradients are obtained by **solving sub-problems** using a first-order optimization method $\mathcal{M}$;
- ideally, $\mathcal{M}$ is **able to adapt to the problem structure** (finite sum, composite regularization).
- replace L-BFGS steps by proximal point steps if no sufficient decrease is estimated $\Rightarrow$ **no line search on** F;

Obtaining inexact gradients

```
Algorithm Procedure ApproxGradient
input Current point $x$ in $\mathbb{R}^d$; smoothing parameter $\kappa > 0$.
1: Compute the approximate mapping using an optimization method $\mathcal{M}$:
       $$z \approx \arg\min_{w \in \mathbb{R}^d} \left\{ h(w) \triangleq f(w) + \frac{\kappa}{2}\|w - x\|^2 \right\},$$
2: Estimate the gradient $\nabla F(x)$
       $$g = \kappa(x - z).$$
output approximate gradient estimate $g$, objective value $F_a \triangleq h(z)$, proximal mapping $z$.
```

```
Algorithm QNing
input $x_0$ in $\mathbb{R}^p$; number of iterations $K$; $\kappa > 0$; minimization algorithm $\mathcal{M}$.
 1: Initialization: $(g_0, F_0, z_0) = \text{ApproxGradient}(x_0, \mathcal{M})$; $B_0 = \kappa I$.
 2: for $k = 0, \ldots, K-1$ do
 3:     Perform the Quasi-Newton step
            $$x_{\text{test}} = x_k - B_k^{-1} g_k$$
            $$(g_{\text{test}}, F_{\text{test}}, z_{\text{test}}) = \text{ApproxGradient}(x_{\text{test}}, \mathcal{M}) \; .$$
 4:     if $F_{\text{test}} \leq F_k - \frac{1}{2\kappa}\|g_k\|^2$, then
 5:         $(x_{k+1}, g_{k+1}, F_{k+1}, z_{k+1}) = (x_{\text{test}}, g_{\text{test}}, F_{\text{test}}, z_{\text{test}})$.
 6:     else
 7:         Update the current iterate with the last proximal mapping:
                $$x_{k+1} = z_k = x_k - (1/\kappa) g_k$$
                $$(g_{k+1}, F_{k+1}, z_{k+1}) = \text{ApproxGradient}(x_{k+1}, \mathcal{M}) \; .$$
 8:     end if
 9:     update $B_{k+1} = \text{L-BFGS}(B_k, x_{k+1} - x_k, g_{k+1} - g_k)$.
10: end for
output last proximal mapping $z_K$ (solution).
```

```
Algorithm QNing
input $x_0$ in $\mathbb{R}^p$; number of iterations $K$; $\kappa > 0$; minimization algorithm $\mathcal{M}$.
1: Initialization: $(g_0, F_0, z_0) = \text{ApproxGradient}(x_0, \mathcal{M})$; $B_0 = \kappa I$.
2: for $k = 0, \ldots, K-1$ do
3:     Perform the Quasi-Newton step
           $x_{\text{test}} = x_k - B_k^{-1} g_k$
           $(g_{\text{test}}, F_{\text{test}}, z_{\text{test}}) = \text{ApproxGradient}(x_{\text{test}}, \mathcal{M})$ .
output last proximal mapping $z_K$ (solution).
```

The main characters:

- the sequence $(x_k)_{k \geq 0}$ that minimizes F;
- the sequence $(z_k)_{k \geq 0}$ produced by $\mathcal{M}$ that minimizes f;
- the gradient approximations $g_k \approx \nabla F(x_k)$;
- the function value approximations $F_k \approx F(x_k)$;
- an L-BFGS update with inexact gradients;
- an approximate sufficient descent condition.

Requirements on $\mathcal{M}$ and restarts

Method $\mathcal{M}$

- Say a sub-problem consists of minimizing h; we want $\mathcal{M}$ to produce a sequence of iterates $(w_t)_{t\geq 0}$ with **linear convergence rate**

$$h(w_t) - h^\star \leq C_{\mathcal{M}}(1-\tau_{\mathcal{M}})^t(h(w_0) - h^\star).$$

Restarts

- When f is smooth, we **initialize** $w_0 = x$ when solving

$$\min_{w\in\mathbb{R}^d} \left\{ f(w) + \frac{\kappa}{2}\|w-x\|^2 \right\}.$$

- When $f = f_0 + \psi$ is composite, we use the initialization

$$w_0 = \arg\min_{w\in\mathbb{R}^d} \left\{ f_0(x) + \langle \nabla f_0(x), w - x\rangle + \frac{L+\kappa}{2}\|w-x\|^2 + \psi(w) \right\}.$$

When do we stop the method $\mathcal{M}$?

Three strategies to balance outer and inner computations

(a) use a **pre-defined sequence** $(\varepsilon_k)_{k\geq 0}$ and stop the optimization method $\mathcal{M}$ when the approximate proximal mapping is ε_k-accurate.

(b) define an **adaptive stopping criterion** that depends on quantities that are available at iteration k.

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$ for solving each sub-problem.

When do we stop the method $\mathcal{M}$?

Three strategies to balance outer and inner computations

(a) use a **pre-defined sequence** $(\varepsilon_k)_{k\geq 0}$ and stop the optimization method $\mathcal{M}$ when the approximate proximal mapping is ε_k-accurate.

(b) define an **adaptive stopping criterion** that depends on quantities that are available at iteration k.

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$ for solving each sub-problem.

Remarks

- We have already seen all of this for Catalyst.

When do we stop the method $\mathcal{M}$?

Three strategies for μ-strongly convex objectives f

(a) use a **pre-defined sequence** $(\varepsilon_k)_{k\geq 0}$ and stop the optimization method $\mathcal{M}$ when the approximate proximal mapping is ε_k-accurate.

$$\varepsilon_k = \frac{1}{2}C(1-\rho)^{k+1} \quad \text{with} \quad C \geq f(x_0) - f^* \;\; \text{and} \;\; \rho = \frac{\mu}{4(\mu+\kappa)}.$$

(b) For minimizing $h(w) = f(w) + (\kappa/2)\|w-x\|^2$, stop when

$$h(w_t) - h^\star \leq \frac{\kappa}{36}\|w_t - x\|^2.$$

(c) use a **pre-defined budget** $T_{\mathcal{M}}$ of iterations of the method $\mathcal{M}$ for solving each sub-problem with

$$T_{\mathcal{M}} = \frac{1}{\tau_{\mathcal{M}}}\log\left(19C_{\mathcal{M}}\frac{L+\kappa}{\kappa}\right).$$ (be more aggressive in practice)

Remarks and worst-case global complexity

Composite objectives and sparsity

Consider a composite problem with a sparse solution (e.g., $\psi = \ell_1$). The method produces two sequences $(x_k)_{k \geq 0}$ and $(z_k)_{k \geq 0}$;

- $F(x_k) \rightarrow F^\star$, minimizes the **smoothed objective** $\Rightarrow$ no sparsity;
- $f(z_k) \rightarrow f^\star$, minimizes the **true objective** $\Rightarrow$ the iterates may be sparse if $\mathcal{M}$ handles composite optimization problems;

Global complexity

The number of iterations of $\mathcal{M}$ to guarantee $f(z_k) - f^\star \leq \varepsilon$ is at most

- $\tilde{O}(\frac{\mu+\kappa}{\tau_{\mathcal{M}}\mu} \log(1/\varepsilon))$ for μ-strongly convex problems.
- $\tilde{O}(\frac{\kappa R^2}{\tau_{\mathcal{M}}\varepsilon})$ for convex problems.

Global Complexity and choice of κ

Example for gradient descent

With the right step-size, we have $\tau_{\mathcal{M}} = (\mu+\kappa)/(L+\kappa)$ and the complexity for $\mu > 0$ becomes

$$\tilde{O}\left(\frac{L+\kappa}{\mu}\log(1/\varepsilon)\right).$$

Example for SVRG for minimizing the sum of n functions

$\tau_{\mathcal{M}} = \min(1/n, (\mu+\kappa)/(L+\kappa))$ and the complexity for $\mu > 0$ is

$$\tilde{O}\left(\max\left(\frac{\mu+\kappa}{\mu}n, \frac{L+\kappa}{\mu}\right)\log(1/\varepsilon)\right).$$

Global Complexity and choice of κ

Example for gradient descent

With the right step-size, we have $\tau_{\mathcal{M}} = (\mu + \kappa)/(L + \kappa)$ and the complexity for $\mu > 0$ becomes

$$\tilde{O}\left(\frac{L+\kappa}{\mu}\log(1/\varepsilon)\right).$$

Example for SVRG for minimizing the sum of n functions

$\tau_{\mathcal{M}} = \min(1/n, (\mu + \kappa)/(L + \kappa))$ and the complexity for $\mu > 0$ is

$$\tilde{O}\left(\max\left(\frac{\mu+\kappa}{\mu}n, \frac{L+\kappa}{\mu}\right)\log(1/\varepsilon)\right).$$

QNing does not provide any theoretical acceleration, but it does not degrade significantly the worst-case performance of $\mathcal{M}$ (unlike L-BFGS vs gradient descent).

Global Complexity and choice of κ

Example for gradient descent

With the right step-size, we have $\tau_{\mathcal{M}} = (\mu+\kappa)/(L+\kappa)$ and the complexity for $\mu > 0$ becomes

$$\tilde{O}\left(\frac{L+\kappa}{\mu}\log(1/\varepsilon)\right).$$

Example for SVRG for minimizing the sum of n functions

$\tau_{\mathcal{M}} = \min(1/n, (\mu+\kappa)/(L+\kappa))$ and the complexity for $\mu > 0$ is

$$\tilde{O}\left(\max\left(\frac{\mu+\kappa}{\mu}n, \frac{L+\kappa}{\mu}\right)\log(1/\varepsilon)\right).$$

Then, how to choose κ?
(i) assume that L-BFGS steps do as well as Nesterov.
(ii) **choose κ as in Catalyst**.

Experiments: formulations

- ℓ_2-**regularized Logistic Regression**:

$$\min_{x \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp(-b_i\, a_i^T x)\right) + \frac{\mu}{2}\|x\|^2,$$

- ℓ_1-**regularized Linear Regression (LASSO)**:

$$\min_{x \in \mathbb{R}^d} \quad \frac{1}{2n} \sum_{i=1}^{n} (b_i - a_i^T x)^2 + \lambda \|x\|_1,$$

- $\ell_1 - \ell_2^2$-**regularized Linear Regression (Elastic-Net)**:

$$\min_{x \in \mathbb{R}^d} \quad \frac{1}{2n} \sum_{i=1}^{n} (b_i - a_i^T x)^2 + \lambda \|x\|_1 + \frac{\mu}{2}\|x\|^2,$$

Experiments: Datasets

We consider four standard machine learning datasets with different characteristics in terms of size and dimension

name	covtype	alpha	real-sim	rcv1
n	581 012	250 000	72 309	781 265
d	54	500	20 958	47 152

- we simulate the ill-conditioned regime $\mu = 1/(100n)$;
- λ for the Lasso leads to about 10% non-zero coefficients.

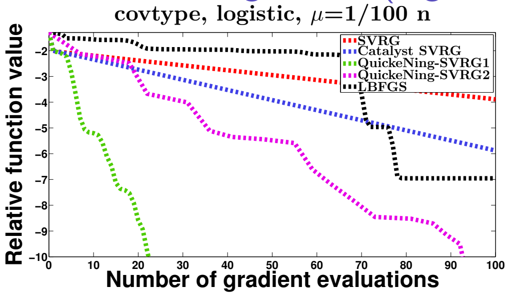
Experiments: QNing-SVRG

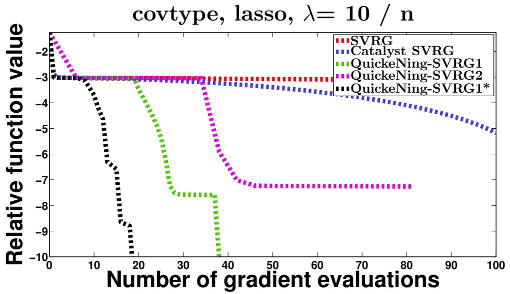
We consider the methods

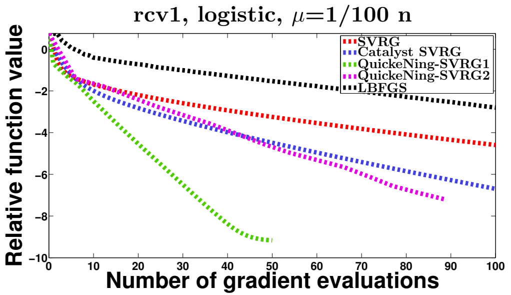
- **SVRG**: the Prox-SVRG algorithm of **?**.
- **Catalyst-SVRG**: Catalyst applied to SVRG;
- **L-BFGS** (for smooth objectives): Mark Schmidt's implementation.
- **QNing-SVRG1**: QNing with aggressive strategy (c): one pass over the data in the inner loop.
- **QNing-SVRG2**: strategy (b), compatible with theory.

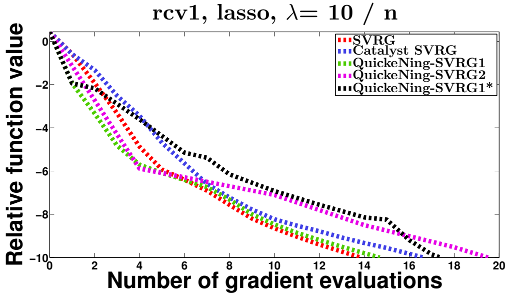
We produce 12 figures (3 formulations, 4 datasets).

Experiments: QNing-SVRG (log scale)







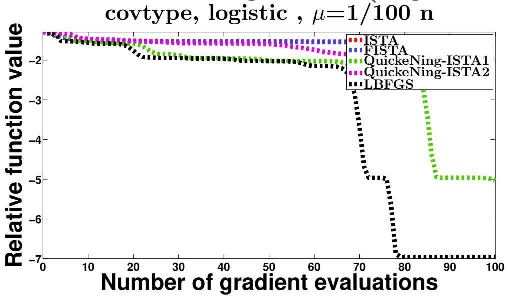


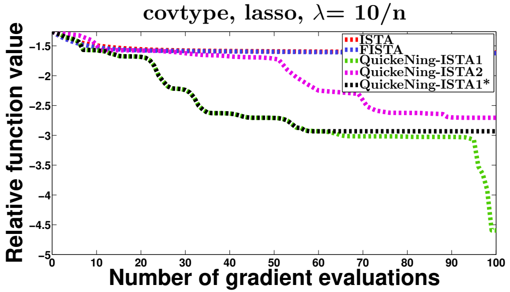
Experiments: QNing-ISTA

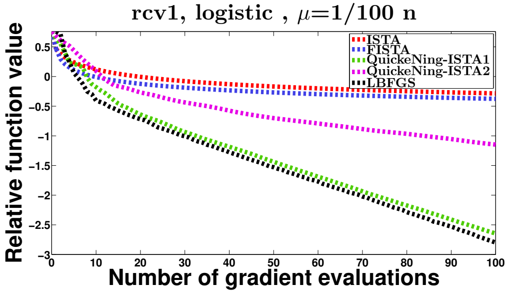
We consider the methods

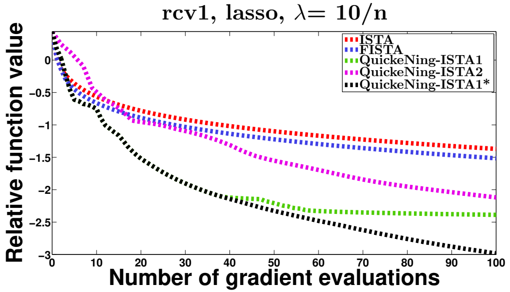
- **ISTA**: the proximal gradient descent method with line search.
- **FISTA**: the accelerated ISTA of **?**.
- **L-BFGS** (for smooth objectives): Mark Schmidt's implementation.
- **QNing-ISTA1**: QNing with aggressive strategy (c): one pass over the data in the inner loop.
- **QNing-ISTA2**: strategy (b), compatible with theory.

Experiments: QNing-ISTA (log scale)

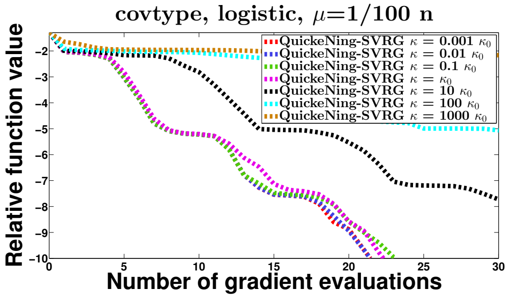


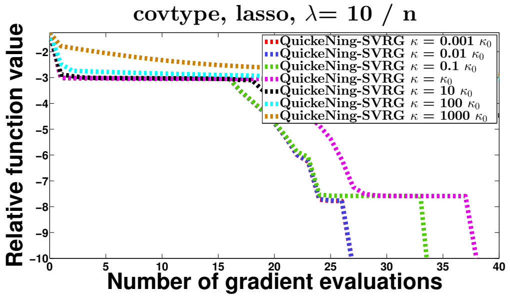


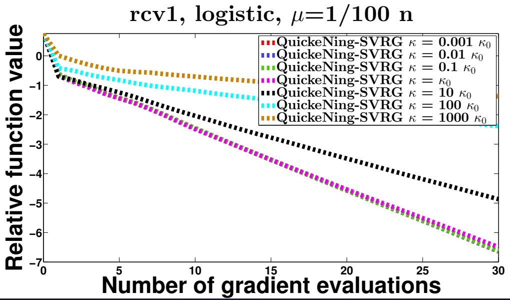


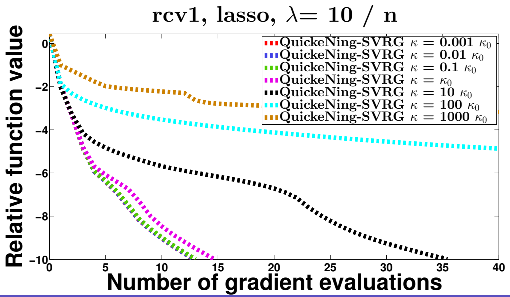


Experiments: Influence of κ

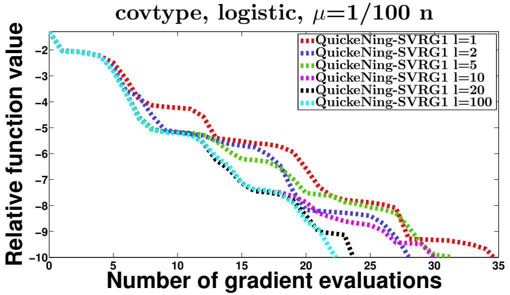


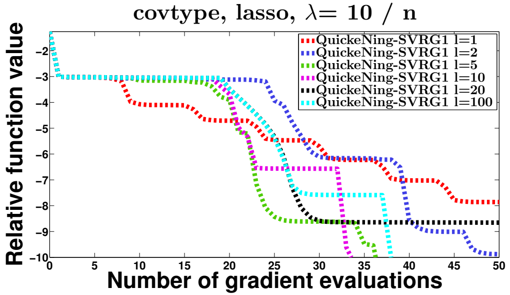


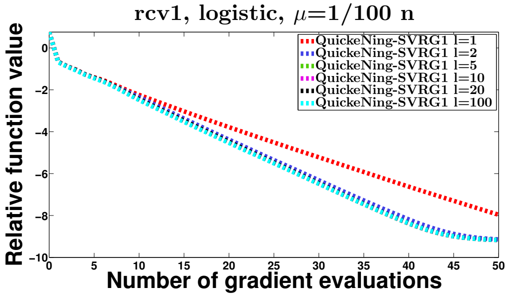


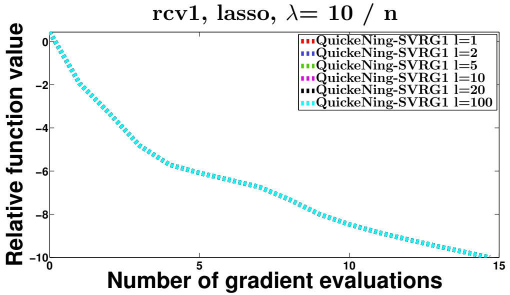


Experiments: Influence of l









Conclusions and perspectives

- A simple generic Quasi-Newton method for composite functions, with simple sub-problems, and complexity guarantees.
- We also have a variant for dual approaches.
- Does not solve the gap between theory and practice for L-BFGS.

Perspectives

- QNing-BCD, QNing-SAG,SAGA,SDCA...
- Other types of smoothing? $\Rightarrow$ Links with recent Quasi-Newton methods applied to other envelopes (**?**).
- Simple line search improves slightly the performance.

Part VII: the Cyanure software package

`http://julien.mairal.org/cyanure/welcome.html`

The Cyanure software package

Binary classification with ℓ_2-logistic regression on the Criteo dataset (21Gb, huge sparse matrix). We use a three-years-old quad-core workstation with 32Gb of memory.

```
import cyanure as cyan
import scipy.sparse
import numpy as np
#load criteo dataset 21Gb, n=45840617, p=999999
dataY=np.load('criteo_y.npz',allow_pickle=True); y=dataY['y']
X = scipy.sparse.load_npz('criteo_X.npz')
#normalize the rows of X in-place, without performing any copy
cyan.preprocess(X,normalize=True,columns=False)
#declare a binary classifier for l2-logistic regression
classifier=cyan.BinaryClassifier(loss='logistic',penalty='l2')
# uses the auto solver by default, performs at most 500 epochs
classifier.fit(X,y,lambd=0.1/X.shape[0],max_epochs=500,tol=1e-3,it0=5)
```

The Cyanure software package

```
Matrix X, n=45840617, p=999999
*********************************
Catalyst Accelerator, MISO Solver, Incremental Solver with uniform sampling
Logistic Loss is used with L2 regularization
Epoch: 5, primal objective: 0.456014, time: 92.5784
Best relative duality gap: 14383.9
Epoch: 10, primal objective: 0.450885, time: 227.593
Best relative duality gap: 1004.69
Epoch: 15, primal objective: 0.450728, time: 367.939
Best relative duality gap: 6.50049
Epoch: 20, primal objective: 0.450724, time: 502.954
Best relative duality gap: 0.068658
Epoch: 25, primal objective: 0.450724, time: 643.323
Best relative duality gap: 0.00173208
Epoch: 30, primal objective: 0.450724, time: 778.363
Best relative duality gap: 0.00173207
Epoch: 35, primal objective: 0.450724, time: 909.426
Best relative duality gap: 9.36947e-05
Time elapsed : 928.114
```

The Cyanure software package

We now learn an SVM with ℓ_1-regularization on this laptop.

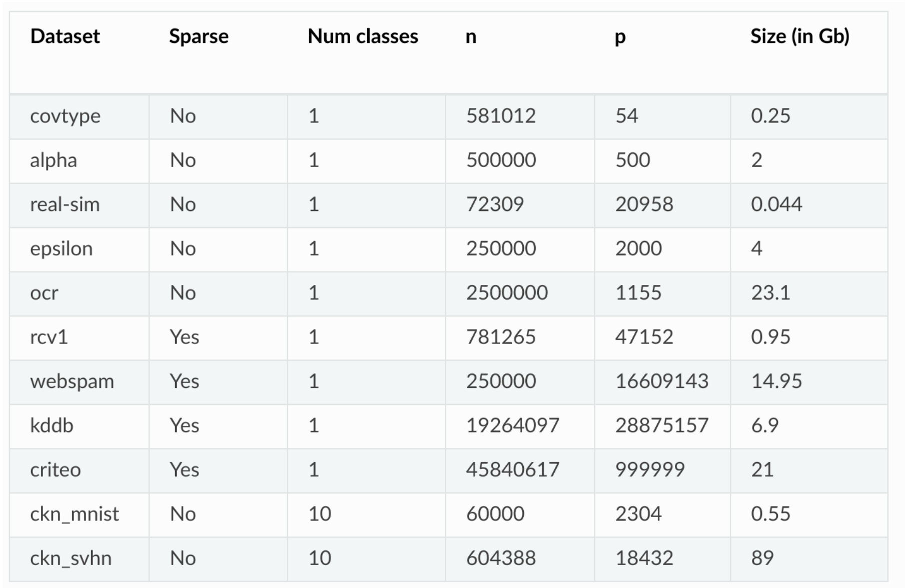
```
import cyanure as cyan
import numpy as np
import scipy.sparse
#load rcv1 dataset about 1Gb, n=781265, p=47152
data = np.load('rcv1.npz',allow_pickle=True); y=data['y']; X=data['X']
X = scipy.sparse.csc_matrix(X.all()).T # n x p matrix, csr format
#normalize the rows of X in-place, without performing any copy
cyan.preprocess(X,normalize=True,columns=False)
#declare a binary classifier for squared hinge loss + l1 regularization
classifier=cyan.BinaryClassifier(loss='sqhinge',penalty='l2')
# uses the auto solver by default, performs at most 500 epochs
classifier.fit(X,y,lambd=0.000005,max_epochs=500,tol=1e-3)
```

The Cyanure software package

```
Matrix X, n=781265, p=47152
Memory parameter: 20
*********************************
QNing Accelerator, MISO Solver
Squared Hinge Loss with L1 regularization
Epoch: 10, primal objective: 0.0915524, time: 7.33038
Best relative duality gap: 0.387338
Epoch: 20, primal objective: 0.0915441, time: 15.524
Best relative duality gap: 0.00426003
Epoch: 30, primal objective: 0.0915441, time: 25.738
Best relative duality gap: 0.000312145
Time elapsed : 26.0225
Total additional line search steps: 8
Total skipping l-bfgs steps: 0
```

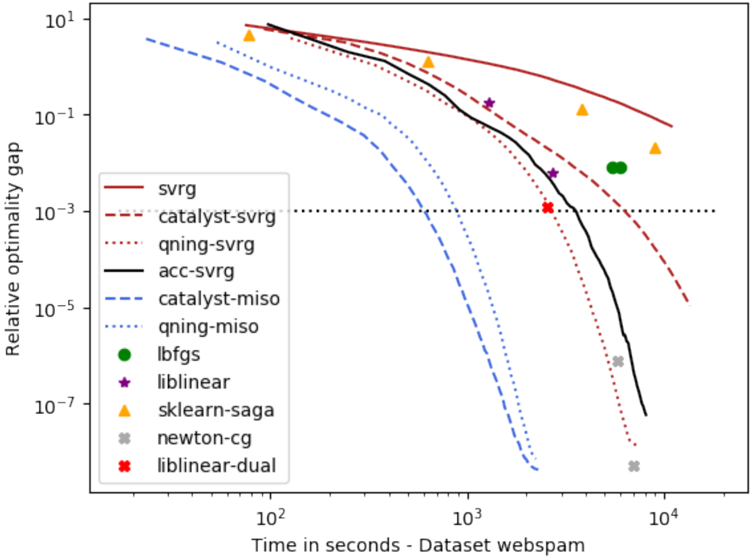
Other examples are available on the website.

The Cyanure software package, benchmarks

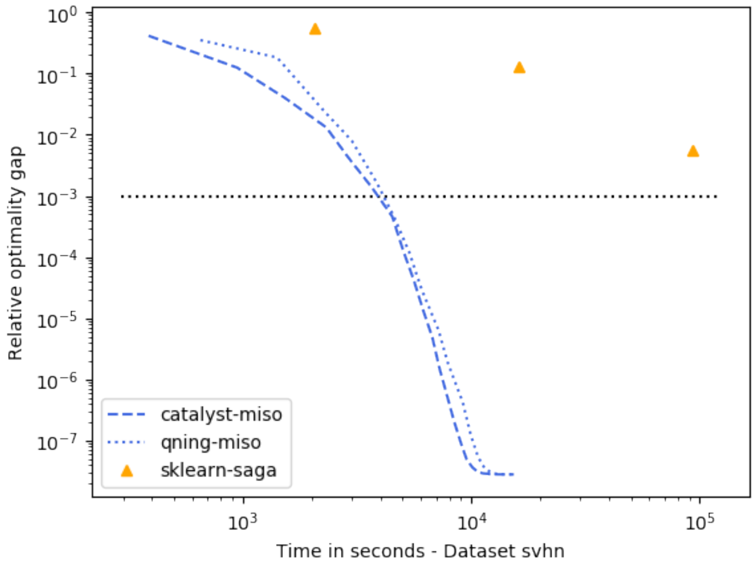


Dataset	Sparse	Num classes	n	p	Size (in Gb)
covtype	No	1	581012	54	0.25
alpha	No	1	500000	500	2
real-sim	No	1	72309	20958	0.044
epsilon	No	1	250000	2000	4
ocr	No	1	2500000	1155	23.1
rcv1	Yes	1	781265	47152	0.95
webspam	Yes	1	250000	16609143	14.95
kddb	Yes	1	19264097	28875157	6.9
criteo	Yes	1	45840617	999999	21
ckn_mnist	No	10	60000	2304	0.55
ckn_svhn	No	10	604388	18432	89

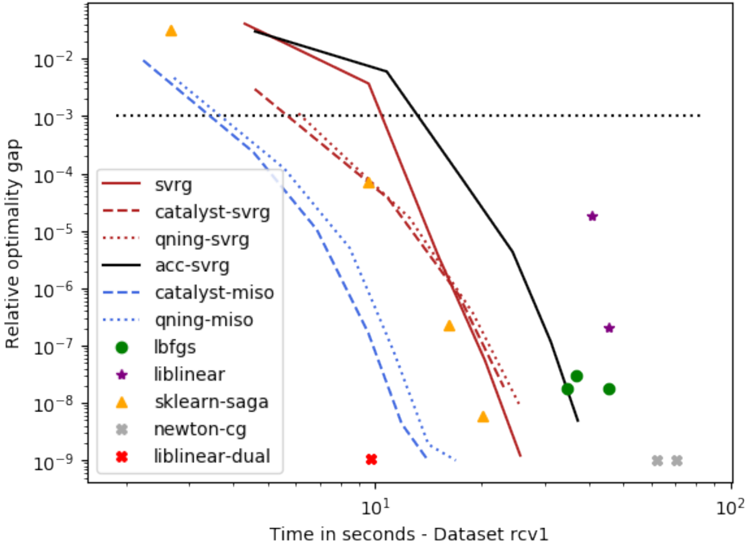
The Cyanure software package, benchmarks



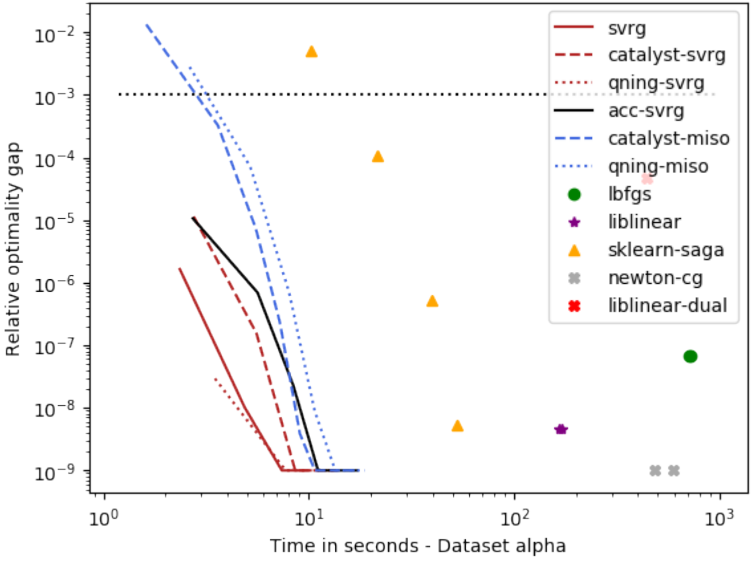
The Cyanure software package, benchmarks



The Cyanure software package, benchmarks



The Cyanure software package, benchmarks



Conclusion

Challenges for algorithms

- going beyond the comfortable convex setting with i.i.d. data.
- better exploit the function curvature for nonconvex problems.

Conclusion

Challenges for algorithms

- going beyond the comfortable convex setting with i.i.d. data.
- better exploit the function curvature for nonconvex problems.

Challenges for Cyanure

Cyanure is still in its early stage. Do not hesitate to post issues/request on github.

Todo list

- Interface for R and Matlab.
- Improve scikit-learn compatibility.
- . . .

Any suggestion is welcome.

References I

A. Agarwal and L. Bottou. A lower bound for the optimization of finite sums. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *arXiv preprint arXiv:1603.05953*, 2016.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009a.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009b.

J.V. Burke and Maijian Qian. On the superlinear convergence of the variable metric proximal point algorithm using Broyden and BFGS matrix secant updating. *Mathematical Programming*, 88(1): 157–181, 2000.

E. J. Candes and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 2005.

References II

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20:33–61, 1999.

Xiaojun Chen and Masao Fukushima. Proximal quasi-Newton methods for nondifferentiable convex optimization. *Mathematical Programming*, 85(2):313–334, 1999.

J. F. Claerbout and F. Muir. Robust modeling with erratic data. *Geophysics*, 38(5):826–844, 1973.

P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *SIAM Multiscale Modeling and Simulation*, 4(4):1168–1200, 2006.

David Corfield, Bernhard Schölkopf, and Vladimir Vapnik. Falsificationism and statistical learning theory: Comparing the popper and vapnik-chervonenkis dimensions. *Journal for General Philosophy of Science*, 40(1):51–58, 2009.

I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11): 1413–1457, 2004.

References III

A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems (NIPS)*, 2014a.

A. J. Defazio, T. S. Caetano, and J. Domke. Finito: A faster, permutable incremental gradient method for big data problems. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014b.

Olivier Devolder, F. Glineur, and Yurii Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, 146(1-2):37–75, 2014.

Aymeric Dieuleveut, Nicolas Flammarion, and Francis Bach. Harder, better, faster, stronger convergence rates for least-squares regression. *Journal of Machine Learning Research*, 18: 101:1–101:51, 2017. URL `http://jmlr.org/papers/v18/papers/v18/16-335.html`.

John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

References IV

Michael P Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

Marc Fuentes, Jérôme Malick, and Claude Lemaréchal. Descentwise inexact proximal algorithms for smooth optimization. *Computational Optimization and Applications*, 53(3):755–769, 2012.

Masao Fukushima and Liqun Qi. A globally and superlinearly convergent algorithm for nonsmooth convex minimization. *SIAM Journal on Optimization*, 6(4):1106–1120, 1996.

S. Ghadimi and G. Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization I: A generic algorithmic framework. *SIAM Journal on Optimization*, 22(4): 1469–1492, 2012.

O. Güler. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2 (4):649–664, 1992.

References V

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Andrei Kulunchakov and Julien Mairal. Estimate sequences for stochastic composite optimization: Variance reduction, acceleration, and robustness to noise. *arXiv preprint arXiv:1901.08788*, 2019.

Guanghui Lan. An optimal randomized incremental gradient method. *arXiv preprint arXiv:1507.02000*, 2015.

Claude Lemaréchal and Claudia Sagastizábal. Practical aspects of the moreau–yosida regularization: Theoretical preliminaries. *SIAM Journal on Optimization*, 7(2):367–385, 1997.

H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, 2015a.

Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2015b.

J. Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855, 2015.

References VI

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

B. Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Revue franÃ§aise d'informatique et de recherche opérationnelle, série rouge*, 1970.

Robert Mifflin. A quasi-second-order proximal bundle algorithm. *Mathematical Programming*, 73(1): 51–72, 1996.

J.J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes-Rendus de l'Académie des Sciences de Paris, Série A, Mathématiques*, 255:2897–2899, 1962.

Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4): 1574–1609, 2009.

References VII

Arkadii Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

Y. Nesterov. *Introductory lectures on convex optimization: a basic course*. Kluwer Academic Publishers, 2004.

Y. Nesterov. Gradient methods for minimizing composite objective function. *Mathematical Programming*, 140(1):125–161, 2013.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

Lam M Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takáč. SGD and Hogwild! convergence without the bounded gradients assumption. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

References VIII

R. D. Nowak and M. A. T. Figueiredo. Fast wavelet-based image deconvolution using the EM algorithm. In *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers.*, 2001.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

P. Paatero and U. Tapper. Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

Saverio Salzo and Silvia Villa. Inexact and accelerated proximal point algorithms. *Journal of Convex Analysis*, 19(4):1167–1192, 2012.

References IX

M. Schmidt, N. Le Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *arXiv:1309.2388*, 2013.

Damien Scieur, Vincent Roulet, Francis Bach, and Alexandre d'Aspremont. Integration methods and optimization algorithms. In *Advances in Neural Information Processing Systems*, pages 1109–1118, 2017.

S. Shalev-Shwartz and T. Zhang. Proximal stochastic dual coordinate ascent. *arXiv:1211.2717*, 2012.

S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, pages 1–41, 2014.

Lorenzo Stella, Andreas Themelis, and Panagiotis Patrinos. Forward-backward quasi-newton methods for nonsmooth optimization problems. *arXiv preprint arXiv:1604.08096*, 2016.

Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518, 2014.

References X

R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.

S.A. van de Geer. ℓ_1-regularization in high-dimensional statistical models. In *Proceedings of the International Congress of Mathematicians*, volume 4, pages 2351–2369, 2010.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1995.

M.J. Wainwright. Sharp thresholds for noisy and high-dimensional recovery of sparsity using ℓ_1-constrained quadratic programming. *IEEE Transactions on Information Theory*, 55(5): 2183–2202, 2009.

Andre Wibisono, Ashia C Wilson, and Michael I Jordan. A variational perspective on accelerated methods in optimization. *proceedings of the National Academy of Sciences*, 113(47): E7351–E7358, 2016.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON convention record*, volume 4, pages 96–104. New York, 1960.

S.J. Wright, R.D. Nowak, and M.A.T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

References XI

D. Wrinch and H. Jeffreys. XLII. On certain fundamental principles of scientific inquiry. *Philosophical Magazine Series 6*, 42(249):369–390, 1921.

L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Wotao Yin, Stanley Osher, Donald Goldfarb, and Jerome Darbon. Bregman iterative algorithms for \ell_1-minimization with applications to compressed sensing. *SIAM Journal on Imaging sciences*, 1 (1):143–168, 2008.

Y. Zhang and L. Xiao. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

P. Zhao and B. Yu. On model selection consistency of Lasso. *Journal of Machine Learning Research*, 7:2541–2563, 2006.