Statistical learning, high dimension and big data

Stéphane Gaïffas

Today is about **optimization for machine learning**. We will learn about the main pillars:

- Proximal gradient descent and acceleration
- Coordinate descent, coordinate gradient descent
- Stochastic gradient descent and beyond

We have seen a lot of problems of the form

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) + g(w)$$

with f a goodness-of-fit function

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \langle w, x_i \rangle)$$

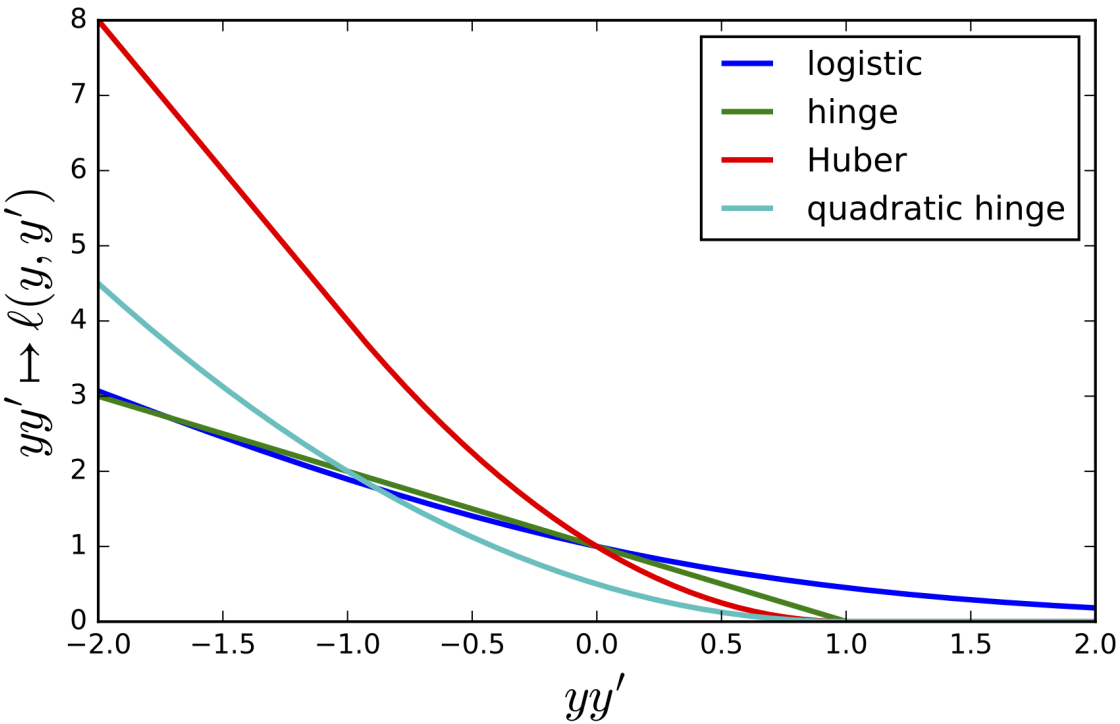
where ℓ is some loss and

$$g(w) = \frac{1}{C} \operatorname{pen}(w)$$

where $\operatorname{pen}(\cdot)$ is some penalization function, examples being $\operatorname{pen}(w) = \frac{1}{2}\|w\|_2^2$ (ridge) and $\operatorname{pen}(w) = \|w\|_1$ (Lasso)

Example of losses for classification

- Logistic loss, $\ell(y, y') = \log(1 + e^{-yy'})$
- Hinge loss, $\ell(y, y') = (1 - yy')_+$
- Quadratic hinge loss, $\ell(y, y') = \frac{1}{2}(1 - yy')_+^2$
- Huber loss $\ell(y, y') = -4yy'\mathbf{1}_{yy' < -1} + (1 - yy')_+^2 \mathbf{1}_{yy' \geq -1}$



Minimization of

$$F(w) = f(w) + g(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \langle x_i, w\rangle) + \frac{1}{C}\,\mathrm{pen}(w)$$

First, note that the gradient and Hessian matrix writes

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \ell'(y_i, \langle x_i, w\rangle) x_i$$

$$\nabla^2 f(w) = \frac{1}{n}\sum_{i=1}^{n} \ell''(y_i, \langle x_i, w\rangle) x_i x_i^\top$$

with

$$\ell'(y, y') = \frac{\partial \ell'(y, y')}{\partial y'} \quad \text{and} \quad \ell''(y, y') = \frac{\partial^2 \ell'(y, y')}{\partial y'^2}$$

And note that f is convex iff

$$y' \mapsto \ell(y_i, y')$$

is for any $i = 1, \dots, n$.

Definition. We say that f is L-**smooth** if it is continuously differentiable and if

$$\|\nabla f(w) - \nabla f(w')\|_2 \leq L\|w - w'\|_2 \quad \text{for any } w, w' \in \mathbb{R}^d$$

If f is twice differentiable, this is equivalent to assuming

$$\lambda_{\max}(\nabla^2 f(w)) \leq L \quad \text{for any } w \in \mathbb{R}^d$$

(largest eigenvalue of the Hessian matrix of f is smaller than L)

For the least-squares loss

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n}(\langle x_i, w\rangle - y_i)x_i, \quad \nabla^2 f(w) = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^\top$$

so that

$$L = \frac{1}{n}\lambda_{\max}\Big(\sum_{i=1}^{n} x_i x_i^\top\Big)$$

For the logit loss

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} y_i (\sigma(y_i \langle x_i, w \rangle) - 1) x_i$$

and

$$\nabla^2 f(w) = \frac{1}{n} \sum_{i=1}^{n} \sigma(y_i \langle x_i, w \rangle)(1 - \sigma(y_i \langle x_i, w \rangle)) x_i x_i^\top$$

so that

$$L = \frac{1}{4n} \lambda_{\max} \Big(\sum_{i=1}^{n} x_i x_i^\top \Big)$$

Gradient descent. Now how to find

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) \quad ?$$

A **key** point: the descent lemma. If f is L-smooth, then

$$f(w') \leq f(w) + \langle \nabla f(w), w' - w \rangle + \frac{L}{2} \|w - w'\|_2^2$$

for any $w, w' \in \mathbb{R}^d$

Proof. Use the fact that

$$\begin{aligned} f(w') &= f(w) + \int_0^1 \langle \nabla f(w + t(w' - w)), w' - w \rangle dt \\ &= f(w) + \langle \nabla f(w), w' - w \rangle \\ &\quad + \int_0^1 \langle \nabla f(w + t(w' - w)) - \nabla f(w), w' - w \rangle dt \end{aligned}$$

So that

$$
\begin{aligned}
&|f(w') - f(w) - \langle \nabla f(w), w' - w \rangle| \\
&\qquad \leq \int_0^1 |\langle \nabla f(w + t(w' - w)) - \nabla f(w), w' - w \rangle dt| \\
&\qquad \leq \int_0^1 \|\nabla f(w + t(w' - w)) - \nabla f(w)\| \|w' - w\| dt \\
&\qquad \leq \int_0^1 Lt\|w' - w\|^2 dt = \frac{L}{2}\|w' - w\|^2
\end{aligned}
$$

which proves the descent lemma. $\square$

It leads, around a point w^k (where k is an iteration counter) to

$$
f(w) \leq f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|_2^2
$$

for any $w \in \mathbb{R}^d$

Remark that

$$\begin{aligned}\operatorname*{argmin}_{w\in\mathbb{R}^d}\left\{f(w^k)+\langle\nabla f(w^k),w-w^k\rangle+\frac{L}{2}\|w-w^k\|_2^2\right\}\\ =\operatorname*{argmin}_{w\in\mathbb{R}^d}\left\|w-\left(w^k-\frac{1}{L}\nabla f(w^k)\right)\right\|_2^2\end{aligned}$$

Hence, it is natural to choose

$$w^{k+1}=w^k-\frac{1}{L}\nabla f(w^k)$$

This is the basic **gradient descent** algorithm

But... where g is gone?

Proximal Gradient descent. Let's put back g:

$$f(w) + g(w) \leq f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|_2^2 + g(w)$$

and again

$$\begin{aligned}
&\operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|_2^2 + g(w) \right\} \\
&\quad = \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ \frac{L}{2} \left\| w - \left(w^k - \frac{1}{L}\nabla f(w^k) \right) \right\|_2^2 + g(w) \right\} \\
&\quad = \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ \frac{1}{2} \left\| w - \left(w^k - \frac{1}{L}\nabla f(w^k) \right) \right\|_2^2 + \frac{1}{L} g(w) \right\} \\
&\quad = \ ????
\end{aligned}$$

Proximal operator. For any $g : \mathbb{R}^d \to \mathbb{R}$ convex, and any $w \in \mathbb{R}^d$, we define

$$\text{prox}_g(w) = \underset{w' \in \mathbb{R}^d}{\text{argmin}} \left\{ \frac{1}{2}\|w - w'\|_2^2 + g(w') \right\}$$

We proved already that if $g(w) = \lambda \|w\|_1$ then

$$\text{prox}_g(w) = S_\lambda(w) = \text{sign}(w) \odot (|w| - \lambda)_+$$

(soft-thresholding) and that if $g(w) = \frac{\lambda}{2}\|w\|_2^2$ then

$$\text{prox}_g(w) = \frac{1}{1+\lambda} w$$

(shrinkage)

Proximal gradient descent (GD)

- **Input**: starting point w^0, Lipschitz constant $L > 0$ for ∇f
- For $k = 1, 2, \ldots$ until *convergence* do

$$w^k \leftarrow \text{prox}_{g/L}\left(w^{k-1} - \frac{1}{L}\nabla f(w^{k-1})\right)$$

- **Return** last w^k

For Lasso

$$w^* \in \underset{w \in \mathbb{R}^d}{\text{argmin}} \left\{ \frac{1}{2n}\|y - Xw\|_2^2 + \lambda \|w\|_1 \right\},$$

the iteration is

$$w^k \leftarrow S_{\lambda/L}\left(w^{k-1} - \frac{1}{Ln}X^\top(Xw^{k-1} - y)\right),$$

where S_λ is the soft-thresholding operator

A theoretical guarantee

- Put for short $F = f + g$,
- Take any $w^* \in \operatorname{argmin}_{w \in \mathbb{R}^d} F(w)$

Theorem. If the sequence $\{w^k\}$ is generated by the proximal gradient descent algorithm, then if f is L-smooth then

$$F(w^k) - F(w^*) \leq \frac{L\|w^0 - w^*\|_2^2}{2k}$$

Comments

- Convergence rate is $O(1/k)$
- ε-accuracy (namely $F(w^k) - F(w^*) \leq \varepsilon$) achieved after $O(L/\varepsilon)$ iterations
- Is it possible to improve the $O(1/k)$ rate? It's very slow!
- Improving this rate **a lot** requires an extra assumption: **strong convexity**

f is μ-**strongly convex** if

$$f(\cdot) - \frac{\mu}{2}\|\cdot\|_2^2$$

is convex. When f if differentiable, it is equivalent to

$$f(w') \geq f(w) + \langle \nabla f(w), w' - w \rangle + \frac{\mu}{2}\|w' - w\|_2^2$$

for any $w, w' \in \mathbb{R}^d$. When f is twice differentiable, this is equivalent to

$$\lambda_{\min}(\nabla^2 f(w)) \geq \mu$$

for any $w \in \mathbb{R}^d$ (smallest eigenvalue of $\nabla^2 f(w)$)

When f is L-smooth, μ-strongly convex and twice differentiable, then

$$\mu \leq \lambda_{\min}(\nabla^2 f(w)) \leq \lambda_{\max}(\nabla^2 f(w)) \leq L$$

for any $w \in \mathbb{R}^d$. We define in this case

$$\kappa = \frac{L}{\mu} \geq 1$$

as the **condition number** of f.

Theorem. If the sequence $\{w^k\}$ is generated by the proximal gradient descent algorithm, and if f is L-smooth and μ-strongly convex, we have

$$F(w^k) - F(w^*) \leq \frac{L}{2} \exp\Big(- \frac{4k}{\kappa + 1} \Big) \|w^0 - w^*\|$$

where $\kappa = L/\mu$ is the condition number of f.

Comments

- Convergence rate is $O(e^{-ck})$
- ε-accuracy achieved after $O(\kappa \log(1/\varepsilon))$ iterations

Acceleration. Can we improve the number of iterations $O(L/\varepsilon)$ (L-smooth) and $O(\frac{L}{\mu}\log(1/\varepsilon))$ (L-smooth and μ strongly-convex) ?

Yes: the idea is to combine w^k and w^{k-1} to find w^{k+1}

Accelerated Proximal Gradient Descent (AGD)

- **Input**: starting points $z^1 = w^0$, Lipschitz constant $L > 0$ for ∇f, $t_1 = 1$
- For $k = 1, 2, \ldots$ until *converged* do

$$w^k \leftarrow \text{prox}_{g/L}(z^k - \frac{1}{L}\nabla f(z^k))$$

$$t_{k+1} \leftarrow \frac{1 + \sqrt{1 + 4t_k^2}}{2}$$

$$z^{k+1} \leftarrow w^k + \frac{t_k - 1}{t_{k+1}}(w^k - w^{k-1})$$

- **Return** last w^k

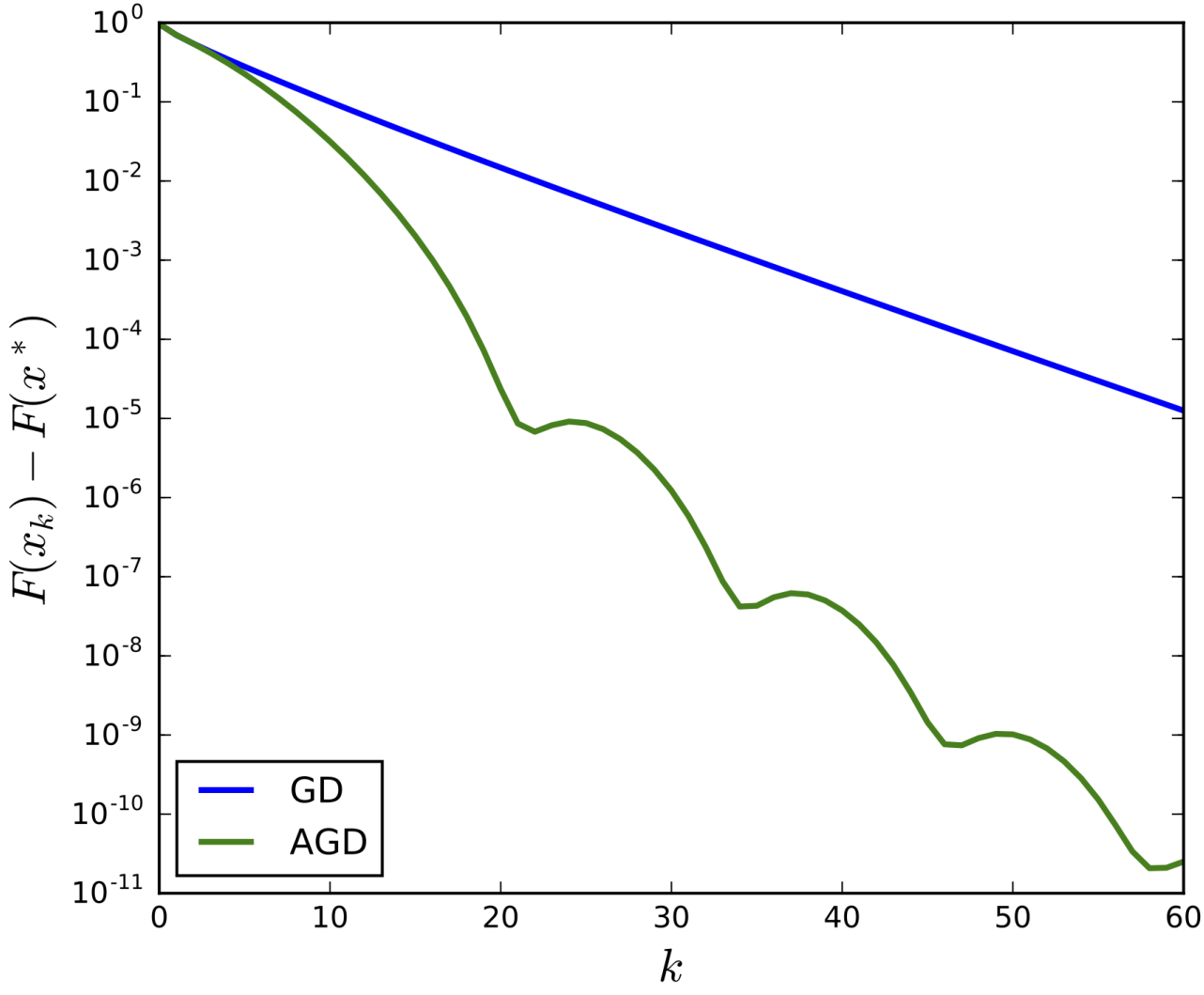
Theorem. Accelerated proximal gradient descent needs

$$O(L/\sqrt{\varepsilon}) \text{ iterations to achieve } \varepsilon\text{-precision}$$

in the L-smooth case and

$$O\left(\sqrt{\frac{L}{\mu}}\log(1/\varepsilon)\right) \text{ iterations to achieve } \varepsilon\text{-precision}$$

in the L-smooth and μ-strongly convex case



Remark. AGD is **not** a descent algorithm, while GD is

Another approach: **coordinate descent**

- Received a lot of attention in machine learning and statistics the last 10 years
- It is state-of-the-art on several machine learning problems, when possible
- This is what is used in many `R` packages and for `scikit-learn` Lasso / Elastic-net and `LinearSVC`

Idea. Minimize one coordinate at a time (keeping all others fixed)

Given $f : \mathbb{R}^d \to \mathbb{R}$ convex and smooth if we have

$$f(w + ze_j) \geq f(w) \text{ for all } z \in \mathbb{R} \text{ and } j = 1, \ldots, d$$

(where $e_j = j$-th canonical vector of $\mathbb{R}^d$) then we have

$$f(w) = \min_{w' \in \mathbb{R}^d} f(w')$$

Proof. $f(w + ze_j) \geq f(w)$ for all $z \in \mathbb{R}$ implies that

$$\frac{\partial f}{\partial w^j}(w) = 0$$

which entails $\nabla f(w) = 0$, so that w is a minimum for f convex and smooth

Exact coordinate descent (CD)

- For $t = 1, \ldots,$
- Choose $j \in \{1, \ldots, d\}$
- Compute

$$w_j^{t+1} = \underset{z \in \mathbb{R}}{\operatorname{argmin}} f(w_1^t, \ldots, w_{j-1}^t, z, w_{j+1}^t, \ldots, w_d^t)$$
$$w_{j'}^{t+1} = w_{j'}^t \quad \text{for } j' \neq j$$

Remarks

- Cycling through the coordinates is arbitrary: uniform sampling, pick a permutation and cycle over it every each d iterations
- Only 1D optimization problems to solve, but a lot of them

Example. Least-squares linear regression

- Let $f(w) = \frac{1}{2n}\|\boldsymbol{X}w - y\|_2^2$
- $\boldsymbol{X}$ features matrix with columns $X^1, \ldots, X^d$
- Minimization over w_j with all other coordinates fixed:

$$0 = \nabla_{w_j} f(w) = \langle X^j, Xw - y\rangle = \langle X^j, X^j w_j + \boldsymbol{X}^{-j} w_{-j} - y\rangle$$

  where X^{-j} is $\boldsymbol{X}$ with j-th columns removed and w_{-j} is w with j-th coordinate removed

- Namely

$$w_j = \frac{\langle X^j, y - \boldsymbol{X}^{-j} w_{-j}\rangle}{\|X^j\|_2^2}$$

- Repeat these updates cycling through the coordinates $j = 1, \ldots, d$

- Namely pick $j \in \{1, \dots, d\}$ at iteration t and do

$$w_j^{t+1} \leftarrow \frac{\langle X^j, y - \boldsymbol{X}^{-j} w_{-j}^t \rangle}{\|X^j\|_2^2}$$
$$w_{j'}^{t+1} \leftarrow w_{j'}^t \quad \text{for } j' \neq j$$

- Written like this, one update complexity is $n \times d$ (matrix-vector product $\boldsymbol{X}^{-j} w_{-j}$ and inner product with X_j)
- Update of all coordinates is $O(nd^2)$? While GD is $O(nd)$ at each iteration...
- No! There is a trick. Defining the current **residual** $r^t \leftarrow y - \boldsymbol{X} w^t$ we can write an update as

$$w_j^{t+1} \leftarrow w_j^t + \frac{\langle X^j, r^t \rangle}{\|X_2^j\|^2} \quad \text{and} \quad r^{t+1} \leftarrow r^t + (w_j^{t+1} - w_j^t) X^j$$

- This is $2n$, which makes the full coordinates update $O(nd)$, like an iteration of GD

Theorem (Warga (1963))
If f is continuously differentiable and strictly convex, then exact coordinate descent converges to a minimum.

Remarks.

- A 1D optimization problem to solve at each iteration: cheap for least-squares, but **can be expensive for other problems**
- Let's solve it approximately, since we have many iterations left
- Replace exact minimization w.r.t. one coordinate by a single gradient step in the 1D problem

Coordinate gradient descent (CGD)

- For $t = 1, \ldots,$
- Choose $j \in \{1, \ldots, d\}$
- Compute

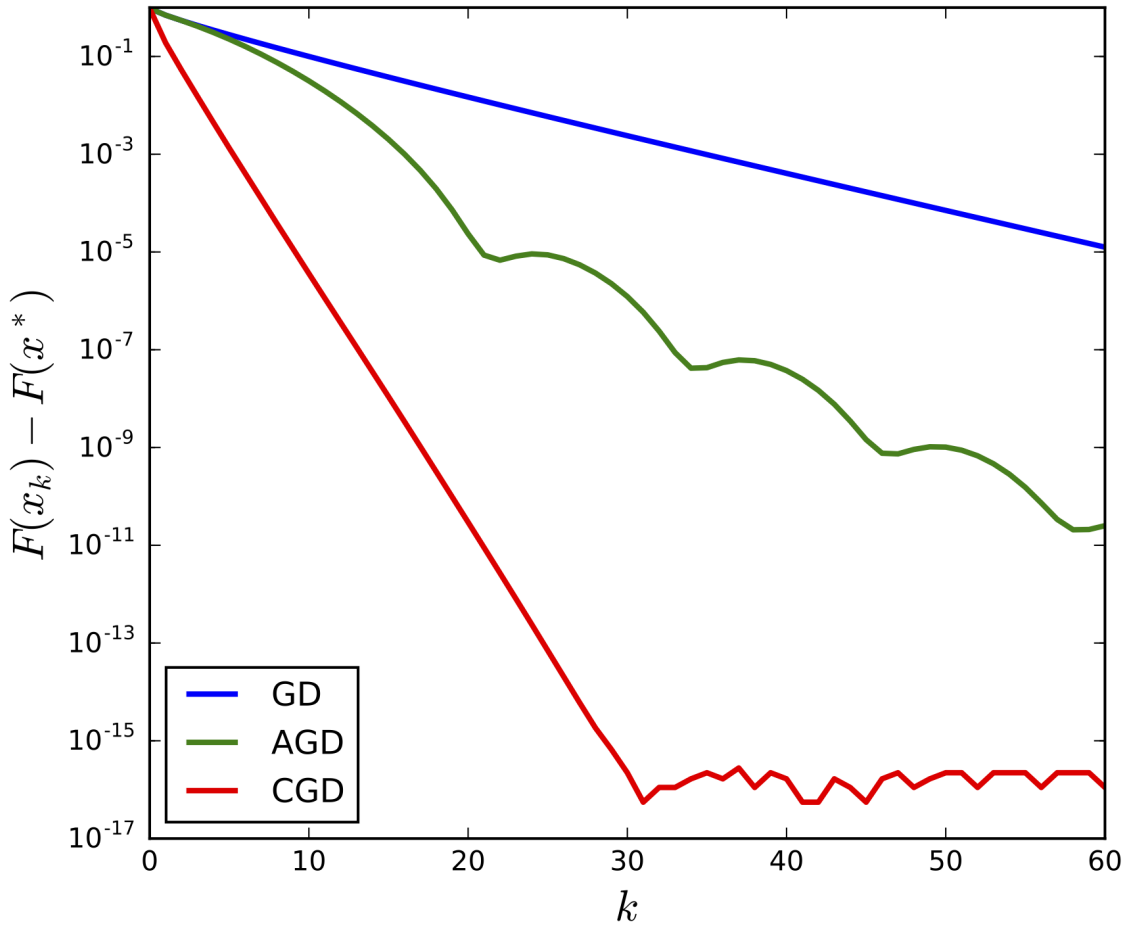
$$w_j^{t+1} = w_j^t - \eta_j \nabla_{w_j} f(w^t)$$
$$w_{j'}^{t+1} = w_{j'}^t \quad \text{for } j' \neq j$$

where

- $\eta_j =$ the step-size for coordinate j, can be taken as $\eta_j = 1/L_j$ where L_j is the Lipchitz constant of

$$f^j(z) = f(w + ze_j) = f(w_1, \ldots, w_{j-1}, z, w_{j+1}, \ldots, w_d)$$

- Cool. Let's try it...



Wow! Coordinate gradient descent is much faster than GD and AGD! But why ?

Theorem (Nesterov (2012)). Assume that f is convex and smooth and that each f^j is L_j-smooth.
Consider a sequence $\{w^t\}$ given by CGD with $\eta_j = 1/L_j$ and coordinates $j_1, j_2, \ldots$ chosen at random: i.i.d and uniform distribution in $\{1, \ldots, d\}$. Then

$$\mathbb{E} f(w^{t+1}) - f(w^*) \leq \frac{n}{n+t}\left((1-\frac{1}{n})(f(w^0)-f(w^*)) + \frac{1}{2}\|w^0 - w^*\|_L^2 \right)$$

with $\|w\|_L^2 = \sum_{j=1}^d L_j w_j^2$.

Remark. Bound in expectation, since coordinates are taken at random. For cycling cordinates $j = (t \mod d) + 1$ the bound is much worse.

Comparison with gradient descent

- GD achieves ε-precision with

$$\frac{L\|w^0 - w^*\|_2^2}{2\varepsilon}$$

  iterations. A single iteration for GD is $O(nd)$
- CGD achieves ε-precision with

$$\frac{d}{\varepsilon}\left((1-\frac{1}{n})(f(w^0)-f(w^*)) + \frac{1}{2}\|w^0 - w^*\|_L^2\right)$$

  iterations. A single iteration for CGD is $O(n)$
- Note that $f(w^0) - f(w^*) \leq \frac{L}{2}\|w^0 - w^*\|_2^2$ but typically $f(w^0) - f(w^*) \ll \frac{L}{2}\|w^0 - w^*\|_2^2$

- So, this is actually

$$\frac{L\|w^0 - w^*\|_2^2}{\varepsilon} \quad \text{against} \quad \frac{1}{\varepsilon}\|w^0 - w^*\|_L^2$$

- Namely L against the L_j
- For least-squares we have $L = \lambda_{\max}(\boldsymbol{X}^\top \boldsymbol{X})$ and $L_j = \|X^j\|_2^2$
- We always have

$$L_j = \|X^j\|_2^2 = \|\boldsymbol{X} e_j\|_2^2 \leq \max_{u:\|u\|_2=1} \|\boldsymbol{X} u\|_2^2 = \lambda_{\max}(\boldsymbol{X}^\top \boldsymbol{X}) = L$$

- And actually it often happens that $L_j \ll L$. For instance, if features are normalized then $L_j = 1$, while $L \approx d$ meaning $L_j = O(L/d)$
- This explains roughly why CGD is much faster than GD for ML problems

- What about non-smooth penalization using CGD ?
- What if I want to use an L1 penalization $g(w) = \lambda \|w\|_1$?
- We only talk about the minimization of $f(w)$ convex and **smooth** using CGD
- What if we want to minimize $f(w) + g(w)$ for g a penalization function, like we did with GD and AGD

Proximal coordinate gradient descent allows to minimize $f(w) + g(w)$ for a **separable** function g, namely a function of the form

$$g(w) = \sum_{j=1}^{d} g_j(w^j)$$

with each g_j convex (eventually not smooth) and such that prox_{g_j} is easy to compute. For Lasso, take $g^j(w^j) = \lambda |w^j|$ for the Lasso (we saw 3 weeks ago that prox_{g_j} is easy to compute)

Proximal coordinate gradient descent (PCGD)

- For $t = 1, \ldots,$
- Choose $j \in \{1, \ldots, d\}$
- Compute

$$w_j^{t+1} \leftarrow \text{prox}_{\eta_j g_j}(w_j^t - \eta_j \nabla_{w_j} f(w^t))$$
$$w_{j'}^{t+1} = w_{j'}^t \quad \text{for } j' \neq j$$

where we recall that

- $\eta_j =$ the step-size for coordinate j, can be taken as $\eta_j = 1/L_j$
- And where $\text{prox}_{\eta_j g_j}$ is

$$\text{prox}_{\eta_j g_j}(w_j) = \underset{z \in \mathbb{R}}{\text{argmin}} \frac{1}{2}(z - w_j)^2 + \eta_j g_j(z)$$

The same Theorem holds as for (CGD) (under the same assumptions, for random draws of coordinates)

Applications to machine learning. Minimization of

$$\min_{w \in \mathbb{R}^d} f(w) + \sum_{j=1}^{d} g_j(w^j)$$

- Regression elastic-net: $f(w) = \frac{1}{2n}\|\boldsymbol{X}w - y\|_2^2$ and $g_j(w) = \lambda(\tau|w_j| + (1-\tau)w_j^2)$
- Logistic regression ℓ_1: $f(w) = \log(1 + \exp(-y \odot \boldsymbol{X}w))$ and $g_j(w) = \lambda|w_j|$
- Box-constrained regression $f(w) = \frac{1}{2n}\|\boldsymbol{X}w - y\|_2^2$ such that $\|w\|_\infty \leq r$
- Non-linear least-squares $f(w) = \frac{1}{2n}\|\boldsymbol{X}w - y\|_2^2$ such that $w_j \geq 0$
- This is what is used in `scikit-learn` for `LinearSVC` when `dual=True` (even if constraint is not separable)

Gradient descent uses iterations

$$w^k \leftarrow w^{k-1} - \eta \nabla f(w^{k-1})$$

and we know that if f is L-smooth then numerical complexity is

$$O(L/\varepsilon)$$

to achieve ε-precision, and if f is also μ-strongly convex then numerical complexity is

$$O\Big(\frac{L}{\mu}\log(1/\varepsilon)\Big)$$

to achieve ε-precision. We should say actually

$$O\Big(n\frac{L}{\mu}\log(1/\varepsilon)\Big)$$

if the "unit " is complexity of $\langle x_i, w\rangle$, namely $O(d)$

We say that these methods are based on **full gradients**, since at each iteration we need to compute

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w),$$

which depends on the whole dataset

Question. If n is large, computing $\nabla f(w)$ is long: need to pass on the whole data before doing a step towards the minimum!

Idea. Large datasets make your modern computer look old: go back to "old" algorithms.

Stochastic gradients

If I choose uniformly at random $I \in \{1, \ldots, n\}$, then

$$\mathbb{E}[\nabla f_I(w)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

$\nabla f_I(w)$ is an **unbiased** but very noisy estimate of the full gradient $\nabla f(w)$

Computation of $\nabla f_I(w)$ only requires the I-th line of data ($O(d)$ and smaller for sparse data, see next)

Stochastic Gradient Descent (SGD)

Input: starting point w^0, steps (learning rates) η_t

For $t = 1, 2, \ldots$ until *convergence* do

- Pick at random (uniformly) i_t in $\{1, \ldots, n\}$
- compute

$$w^t = w^{t-1} - \eta_t \nabla f_{i_t}(w^{t-1})$$

Return last w^t

Remarks

- Each iteration has complexity $O(d)$ instead of $O(nd)$ for full gradient methods
- Possible to reduce this to $O(s)$ when features are s-sparse using **lazy-updates** (more on this later)

Full gradient descent

$$w^k \leftarrow w^{t-1} - \frac{\eta_t}{n} \sum_{i=1}^{n} \nabla f_i(w^{t-1})$$

has $O(nd)$ iteration: numerical complexity $O(n\frac{L}{\mu}\log(\frac{1}{\varepsilon})))$

Stochastic gradient descent

$$w^t \leftarrow w^{t-1} - \eta_t \nabla f_{i_t}(w^{t-1})$$

$O(d)$ iteration: numerical complexity $O(\frac{1}{\mu\varepsilon})$ (more next...)

(both when f is μ-strongly convex and L-smooth)

It does not depend on n for SGD !

Now w^t is a stochastic sequence, that depends on random draws of indices $i_1, \ldots, i_t$, denoted $\mathcal{F}_t$

If i_t is chosen uniformly at random in $\{1, \ldots, n\}$ and independent of previous $\mathcal{F}_{t-1}$ then

$$\mathbb{E}[\nabla f_i(w^{t-1})|\mathcal{F}_{t-1}] = \frac{1}{n}\sum_{i'=1}^{n} \nabla f_{i'}(w^{t-1}) = \nabla f(w^{t-1})$$

SGD uses very noisy unbiased estimations of the full gradient

Polyak-Ruppert averaging: use SGD iterates w^t but return

$$\bar{w}^t = \frac{1}{t}\sum_{t'=1}^{t} w^{t'}$$

Theoretical properties on SGD. If:

- f is convex
- gradients are bounded: $\|\nabla f_i(w)\|_2 \leq b$

we have a convergence rate

$$O\Big(\frac{1}{\sqrt{t}}\Big) \quad \text{with} \quad \eta_t = O\Big(\frac{1}{\sqrt{t}}\Big)$$

and if moreover

- f is μ-strongly convex

the rate is

$$O\Big(\frac{1}{\mu t}\Big) \quad \text{with} \quad \eta_t = O\Big(\frac{1}{\mu t}\Big)$$

Both achieved by ASGD (average SGD)

Under strong convexity, GD versus SGD is

$$O\Big(\frac{n}{\mu}\log\big(\frac{1}{\varepsilon}\big)\Big) \quad \text{versus} \quad O\Big(\frac{1}{\mu\varepsilon}\Big)$$

GD leads to a more accurate solution, but what if n is very large?

Recipe

- SGD is extremely fast in the early iterations (first two passes on the data)
- But it fails to converge accurately to the minimum

Lazy updates.

- Feature vectors can be very sparse (bag-of-words, etc.)
- Complexity of the iteration can reduced from $O(d)$ to $O(s)$, where s is the sparsity of the features.

Typically $d \approx 10^7$ and $s \approx 10^3$

For minimizing

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \langle x_i, w \rangle) + \frac{\lambda}{2}\|w\|_2^2$$

an iteration of SGD writes

$$w^t = (1 - \eta_t \lambda) w^{t-1} - \eta_t \ell'(y_i, \langle x_i, w^{t-1} \rangle) x_i$$

If x_i is s sparse, then computing $\eta_t \ell'(y_i, \langle x_i, w^{t-1} \rangle) x_i$ is $O(s)$, but $(1 - \eta_t \lambda) w^{t-1}$ is $O(d)$

Lazy updates trick.
Put $w^t = s_t \beta^t$, with $s_t \in [0, 1]$ and $s_t = (1 - \eta_t \lambda) s_{t-1}$

$$w^t = (1 - \eta_t \lambda) w^{t-1} - \eta_t \ell'(y_i, \langle x_i, w^{t-1} \rangle) x_i$$

becomes

$$\begin{aligned} s_t \beta^t &= (1 - \eta_t \lambda) s_{t-1} \beta^{t-1} - \eta_t \ell'(y_i, s_{t-1} \langle x_i, \beta^{t-1} \rangle) x_i \\ &= s_t \beta^{t-1} - \eta_t \ell'(y_i, s_{t-1} \langle x_i, \beta^{t-1} \rangle) x_i \end{aligned}$$

so the iteration is now

$$\beta^t = \beta^{t-1} - \frac{\eta_t}{s_t} \ell'(y_i, s_{t-1} \langle x_i, \beta^{t-1} \rangle) x_i$$

which has complexity $O(s)$.

Beyond SGD

Recent results improve this:

- Bottou and LeCun (2005)
- Shalev-Shwartz et al (2007, 2009)
- Nesterov et al. (2008, 2009)
- Bach et al. (2011, 2012, 2014, 2015)
- T. Zhang et al. (2014, 2015)

The problem

- Put $X = \nabla f_I(w)$ with I uniformly chosen at random in $\{1, \ldots, n\}$
- In SGD we use $X = \nabla f_I(w)$ as an approximation of $\mathbb{E}X = \nabla f(w)$
- How to reduce var X ?

An idea

- Reduce it by finding C s.t. $\mathbb{E}C$ is "easy" to compute and such that C is highly correlated with X
- Put $Z_\alpha = \alpha(X - C) + \mathbb{E}C$ for $\alpha \in [0, 1]$. We have

$$\mathbb{E}Z_\alpha = \alpha\mathbb{E}X + (1 - \alpha)\mathbb{E}C$$

and

$$\operatorname{var} Z_\alpha = \alpha^2(\operatorname{var} X + \operatorname{var} C - 2\operatorname{cov}(X, C))$$

- Standard variance reduction: $\alpha = 1$, so that $\mathbb{E}Z_\alpha = \mathbb{E}X$ (unbiased)

Variance reduction of the gradient

In the iterations of SGD, replace $\nabla f_{i_t}(w^{t-1})$ by

$$\alpha(\nabla f_{i_t}(w^{t-1}) - \nabla f_{i_t}(\widetilde{w})) + \nabla f(\widetilde{w})$$

where $\widetilde{w}$ is an "old" value of the iterate, namely use

$$w^t \leftarrow w^{t-1} - \eta\big(\alpha(\nabla f_{i_t}(w^{t-1}) - \nabla f_{i_t}(\widetilde{w})) + \nabla f(\widetilde{w})\big)$$

Several cases

- $\alpha = 1/n$: SAG (Bach et al. 2013)
- $\alpha = 1$: SVRG (T. Zhang et al. 2015, 2015)
- $\alpha = 1$: SAGA (Bach et al., 2014)

Stochastic Average Gradient

Input: starting point w^0, learning rate $\eta > 0$
For $t = 1, 2, \ldots$ until *convergence* do

- Pick uniformly at random i_t in $\{1, \ldots, n\}$
- Put

$$g_t(i) = \begin{cases} \nabla f_i(w^{t-1}) & \text{if } i = i_t \\ g_{t-1}(i) & \text{otherwise} \end{cases}$$

  and compute

$$w^t = w^{t-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_t(i)$$

Return last w^t

Stochastic Variance Reduced Gradient

Input: starting point w^0, learning rate $\eta > 0$
Put $\widetilde{w}_1 \leftarrow w^0$
For $k = 1, 2, \ldots$ until *convergence* do

- Put $w_k^0 \leftarrow \widetilde{w}_k$
- Compute $\nabla f(\widetilde{w}_k)$
- For $t = 0, \ldots, m-1$
  - Pick uniformly at random i in $\{1, \ldots, n\}$
  - Apply the step

$$w_k^{t+1} \leftarrow w_k^t - \eta(\nabla f_i(w_k^t) - \nabla f_i(\widetilde{w}_k) + \nabla f(\widetilde{w}_k))$$

- Set

$$\widetilde{w}_k \leftarrow \frac{1}{m} \sum_{t=1}^{m} w_k^t$$

Return last w_k^t

SAGA

Input: starting point w^0, learning rate $\eta > 0$
Compute $g_0(i) \leftarrow \nabla f_i(w^0)$ for all $i = 1, \ldots, n$
For $t = 1, 2, \ldots$ until *convergence* do

- Pick uniformly at random i_t in $\{1, \ldots, n\}$
- Compute $\nabla f_{i_t}(w^{t-1})$
- Apply

$$w^t \leftarrow w^{t-1} - \eta\left(\nabla f_{i_t}(w^{t-1}) - g_{t-1}(i_t) + \frac{1}{n}\sum_{i=1}^{n} g_{t-1}(i)\right)$$

- Store $g_t(i_t) \leftarrow \nabla f_{i_t}(w^{t-1})$

Return last w^t

Stochastic Variance Reduced Gradient

Phase size typically chosen as $m = n$ or $m = 2n$

If $F = f + g$ with g prox-capable, use

$$w_k^{t+1} \leftarrow \text{prox}_{\eta g}(w_k^t - \eta(\nabla f_i(w_k^t) - \nabla f_i(\widetilde{w}_k) + \nabla f(\widetilde{w}^k)))$$

SAGA

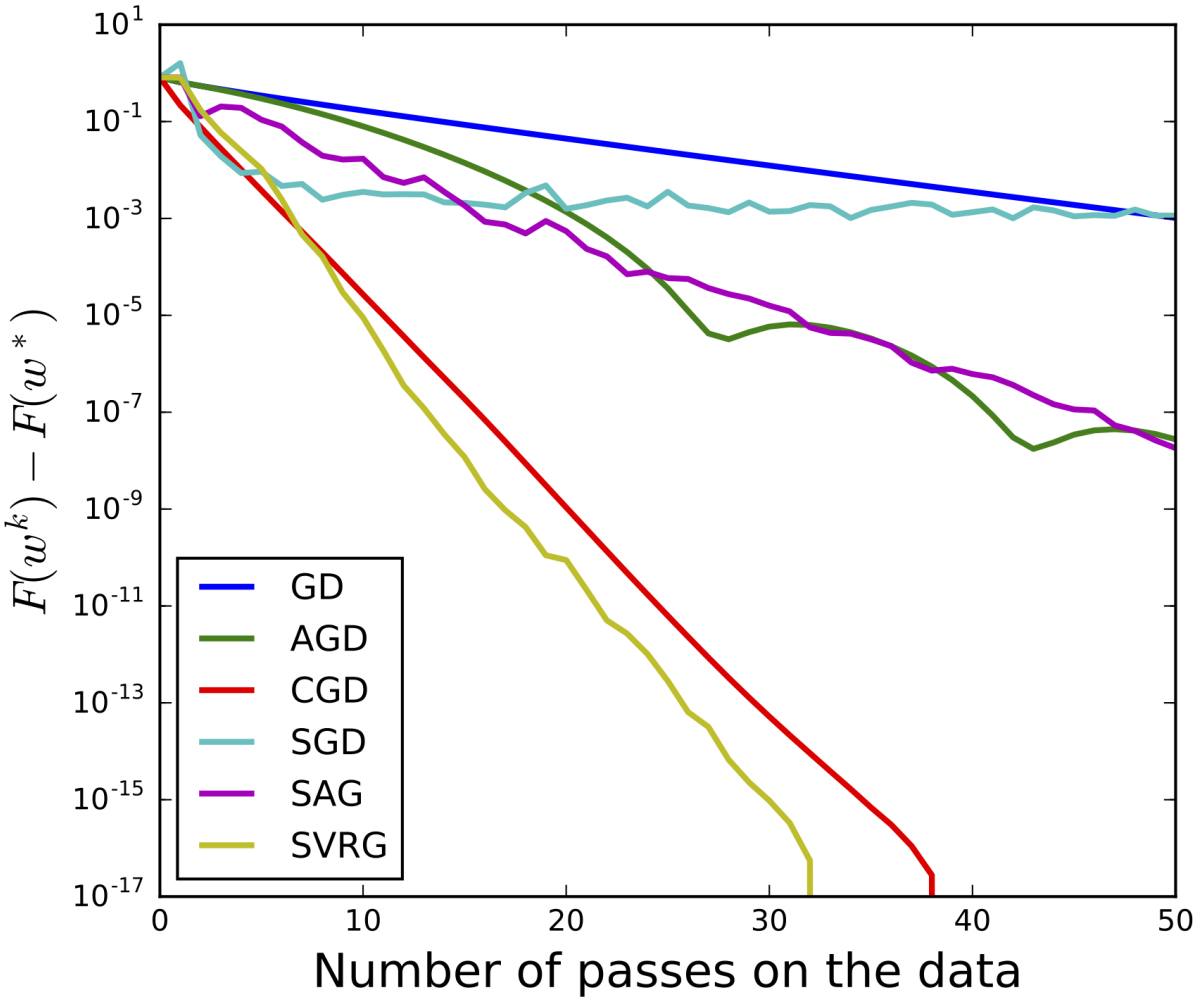
If $F = f + g$ with g prox-capable, use

$$w^t \leftarrow \text{prox}_{\eta g}\left(w^{t-1} - \eta\left(\nabla f_{i_t}(w^{t-1}) - g_{t-1}(i_t) + \frac{1}{n}\sum_{i=1}^{n} g_{t-1}(i)\right)\right)$$

Important remark

- In these algorithms, the step-size η is kept **constant**
- Leads to **linearly convergent algorithms**, with a numerical complexity comparable to SGD!

Algorithms comparison



Don't worry **You** will code all of it this :)

Theoretical guarantees

- Each f_i is L_i-smooth. Put $L_{\max} = \max_{i=1,\dots n} L_i$
- f is μ-strongly convex

For SAG

Take $\eta = 1/(16 L_{\max})$ constant

$$\mathbb{E} f(w^t) - f(w^*) \leq O\Big(\frac{1}{n\mu} + \frac{L_{\max}}{n}\Big) \exp\Big(- t\Big(\frac{1}{8n} \wedge \frac{\mu}{16 L_{\max}}\Big)\Big)$$

The rate is typically faster than gradient descent!

For SVRG

Take η and m such that

$$\rho = \frac{1}{1 - 2\eta L_{\max}} \left(\frac{1}{m\eta\mu} + 2L_{\max}\eta \right) < 1$$

Then

$$\mathbb{E} f(w^k) - f(w^*) \leq \rho^k (f(w^0) - f(w^*))$$

In practice $m = n$ and $\eta = 1/L_{\max}$ works

In summary, about variance reduction

- Complexity $O(d)$ instead of $O(nd)$ at each iteration
- Choice of a **fixed** step-size $\eta > 0$ possible
- Much faster than full gradient descent!

Numerical complexities

- $O(nL/\mu \log(1/\varepsilon))$ for GD
- $O(1/(\mu n))$ for SGD
- $O((n + L_{\max}/\mu) \log(1/\varepsilon))$ for SGD with variance reduction (SAG, SAGA, SVRG, etc.)

where $L =$ Lipschitz constant of $\frac{1}{n}\sum_{i=1}^{n} f_i$.
Note that typically

$$n\frac{L}{\mu}\log(1/\varepsilon) \gg \left(n + \frac{L_{\max}}{\mu}\right)\log(1/\varepsilon)$$

Some practical aspects

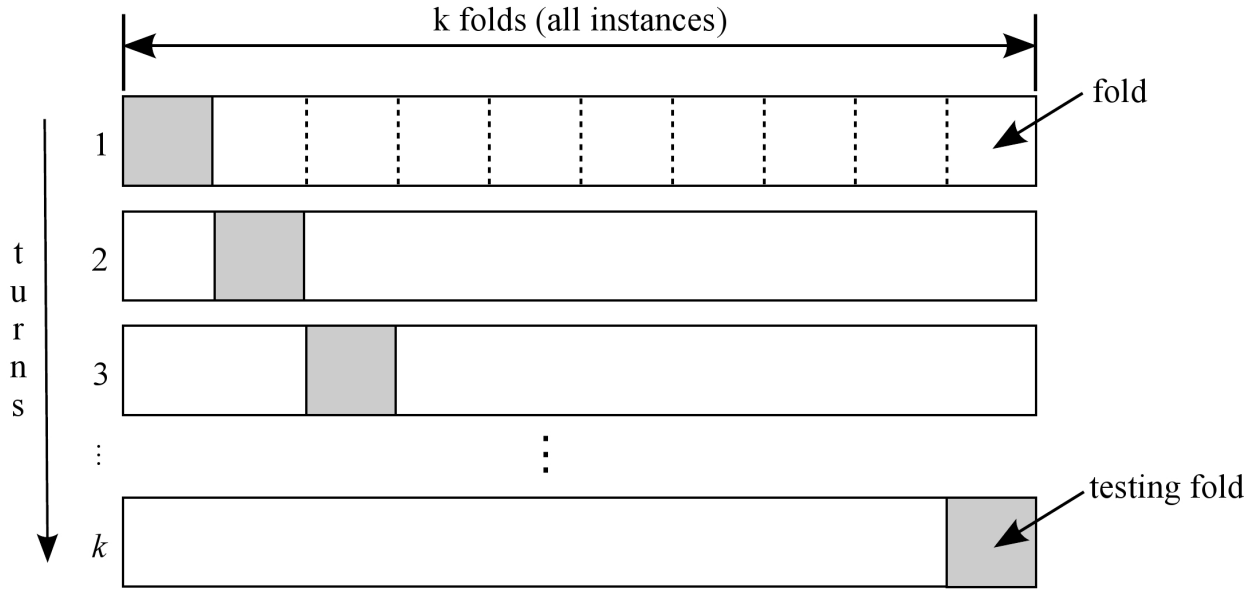
- SAG and SAGA requires extra memory: need to save all the previous gradients!
- Actually no...

$$\nabla f_i(w) = \ell'(y_i, \langle x_i, w \rangle) x_i,$$

  so only need to save $\ell'(y_i, \langle x_i, w \rangle)$
- Memory footprint is $O(n)$ instead of $O(nd)$. If $n = 10^7$, this is 76 Mo
- Can use same lazy updating tricks as for SGD from before

Some practical aspects

- V-fold cross-validation
- Take $V = 5$ or $V = 10$. Pick a random partition $I_1, \ldots, I_V$ of $\{1, \ldots, n\}$, where $|I_v| \approx \frac{n}{V}$ for any $v = 1, \ldots, V$



How to do it with SGD type algorithms?

Some practical aspects

- V-fold cross-validation

Simple solution

When picking a line i at random in the optimization loop, its fold number is given by $i\%V$

- Pick i uniformly at random in $\{1, \ldots, n\}$
- Put $v = i\%V$
- For $v' = 1, \ldots, V$ with $v' \neq v$: update $\hat{w}^{(v')}$ using line i
- Update the testing error of $\hat{w}^{(v)}$ using line i

Some practical aspects

We want to minimize a sequence of objectives

$$f(w) + \lambda g(w)$$

for $\lambda = \lambda_1, \ldots, \lambda_M$, and select the best using V-fold cross-validation

Idea

Use the fact that solutions $\hat{w}^{\lambda_{j-1}}$ and $\hat{w}^{\lambda_j}$ are close when λ_{j-1} and λ_j are

Warm-starting

Put $w^0 = 0$ (I don't know where to start)

For $m = M, \ldots, 1$

- Put $\lambda = \lambda_m$
- Solve the problems starting at x_0 for this value of λ (on each fold)
- Keep the solutions $\hat{w}$ (test it, save it...)
- Put $w^0 \leftarrow \hat{w}$

This allows to solve much more rapidly the sequence of problems

Thank you!