Machine learning

Claire Boyer

October 14th, 2020

Today

1. Complexity, Selection and Penalization
2. Convex analysis: reminder
3. Gradient descent
4. Proximal Gradient descent
5. Acceleration
6. Newton method
7. Coordinate descent
   - Exact coordinate descent
   - Coordinate gradient descent
   - Proximal coordinate gradient descent
8. Stochastic gradient descent

Mainly taken from Stéphane Gaïffas's lectures

Some references :

- Yurii Nesterov, Introductory lectures on convex optimization, *Springer*
- Stephen Boyd and Lieven Vandenberghe, Convex optimization, *Cambridge University Press*
- Lieven Vandenberghe's lectures
- Sébastien Bubeck, Convex Optimization: Algorithms and Complexity
- + research papers

Summary

Goal

Machine Learning

- Learn a rule to construct a predictor $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. the risk $\mathcal{R}(\widehat{f})$ is small on average or with high probability with respect to $\mathcal{D}_n$.

Canonical example: Empirical Risk Minimizer

- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \operatorname{argmin}_{f_\theta, \theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\mathbf{X}_i))$$

- Examples:
  - Linear regression
  - Linear discrimination with

$$\mathcal{S} = \{\mathbf{x} \mapsto \mathtt{sign}\{\beta^T \mathbf{x} + \beta_0\} \,/ \beta \in \mathbb{R}^d, \beta_0 \in \mathbb{R}\}$$

Linear Discrimination

Linear Classifier

- Classifier family:

$$\mathcal{S} = \{ f_\theta : \mathbf{x} \mapsto \mathtt{sign}\{\beta^T \mathbf{x} + \beta_0\} \,/ \beta \in \mathbb{R}^d, \beta_0 \in \mathbb{R} \}$$

- Natural loss: $\ell^{0/1}(Y, f(x)) = 1_{y \neq f(x)}$

Empirical Risk Minimization

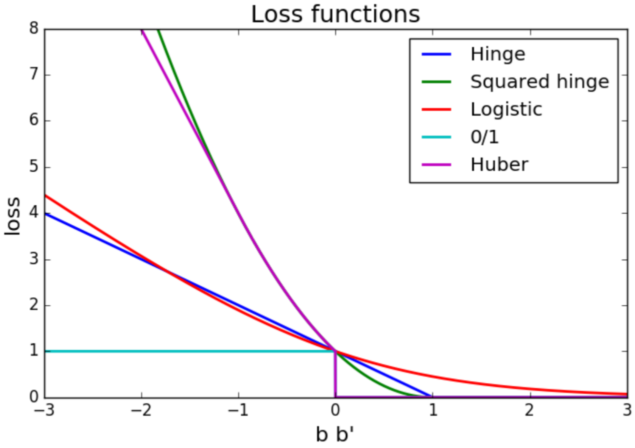
- ERM Classifier:

$$\widehat{f} = f_{\widehat{\theta}} = \operatorname{argmin}_{f_\theta, \theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} 1_{Y_i \neq f_\theta(\mathbf{x}_i))}$$

- Not smooth or convex $\implies$ no easy minimization scheme!
- $\neq$ regression with quadratic loss case!

- How to go beyond?

Different types of losses



Constraint and Penalization

Constrained Optimization

- Choose a constant C.
- Compute β as

$$\operatorname{argmin}_{\beta \in \mathbb{R}^d, \|\beta\|_p \leqslant C} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i(\beta^t x_i)})$$

Lagrangian Reformulation

- Choose λ and compute β as

$$\operatorname{argmin}_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i(\beta^t x_i)}) + \lambda \|\beta\|_p^{p'}$$

  with $p' = p$ except if $p = 0$ where $p' = 1$.
- Easier calibration...

Penalization

Penalized Likelihood

- Minimization of

$$\operatorname{argmin}_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i(\beta^t x_i)}) + \mathrm{pen}(\beta)$$

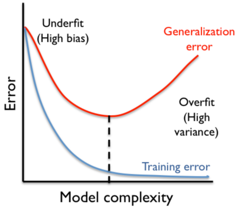
  where $\mathrm{pen}(\beta)$ is a (sparsity promoting) penalty
- Variable selection if β is sparse.

Classical Penalties

- AIC: $\mathrm{pen}(\beta) = \lambda \|\beta\|_0$ (non convex / sparsity)
- Ridge: $\mathrm{pen}(\beta) = \lambda \|\beta\|_2^2$ (convex / no sparsity)
- Lasso: $\mathrm{pen}(\beta) = \lambda \|\beta\|_1$ (convex / sparsity)
- Elastic net: $\mathrm{pen}(\beta) = \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2$ (convex / sparsity)

- Easy optimization if pen (and the loss) is convex...
- **Need to specify λ!**

Regularization Parameter Issue



- Need to choose λ from the data!

Error behaviour

- Learning/training error (error made on the learning/training set) decays when the regularization parameter decreases.
- Quite different behavior when the error is computed on new observations (generalization error).

- Overfit for complex models: parameters learned are too specific to the learning set!
- General situation! (Think of polynomial fit...)
- Need another criterion than the training error!

Cross Validation

- **Very simple idea:** use a second learning/verification set to compute a verification error.
- Sufficient to avoid over-fitting!

Cross Validation

- Use $\frac{V-1}{V}n$ observations to train and $\frac{1}{V}n$ to verify!
- Validation for a learning set of size $(1-\frac{1}{V}) \times n$ instead of n!

- Most classical variations:
  - Leave One Out,
  - V-fold cross validation.
- Accuracy/Speed tradeoff: $V = 5$ or $V = 10$!

Penalization and Cross-Validation

Practical Selection Methodology

- Choose a penalty shape $\widetilde{\mathrm{pen}}(\beta)$.
- Compute a CV error for a penalty $\lambda\widetilde{\mathrm{pen}}(\beta)$ for all $\lambda \in \Lambda$.
- Determine $\widehat{\lambda}$ the λ minimizing the CV error.
- Compute the final logistic regression with a penalty $\widehat{\lambda}\widetilde{\mathrm{pen}}(\beta)$.

Why not using only CV?

- **If** the penalized likelihood minimization is easy, much cheaper to compute the CV error for all $\lambda \in \Lambda$ than for all $\beta \in \mathbb{R}^d$!
- CV performs best when the set of candidates is not too big (or is structured...)

The problem of the day

We encountered 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

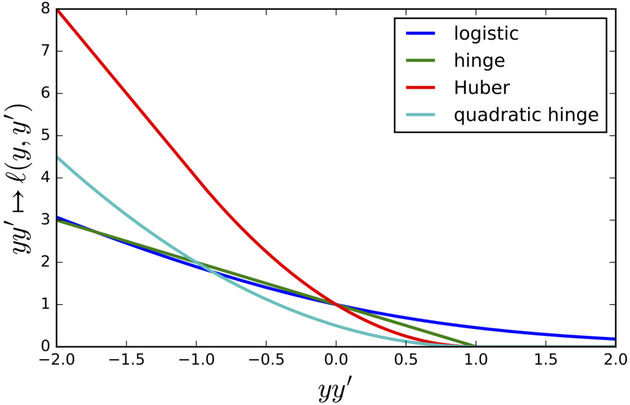
$$g(w) = \frac{1}{C} \operatorname{pen}(w)$$

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)

Examples

Classical 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 $\ell(y, y') = -4yy' \mathbb{1}_{yy' < -1} + (1 - yy')_+^2 \mathbb{1}_{yy' \geqslant -1}$



More generally w/ a differentiable loss

Minimization of

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

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

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

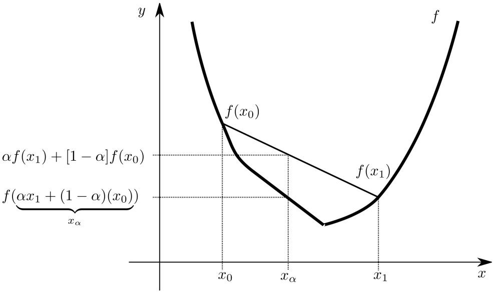
Summary

Definition

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for all $(x, y) \in \mathbb{R}^d$ and all $\alpha \in [0, 1]$

$$f(\alpha x + (1 - \alpha)y) \leqslant \alpha f(x) + (1 - \alpha)f(y)$$

Interpretation : The graph of a cvx fct is always below the segment joining 2 points on the graph.

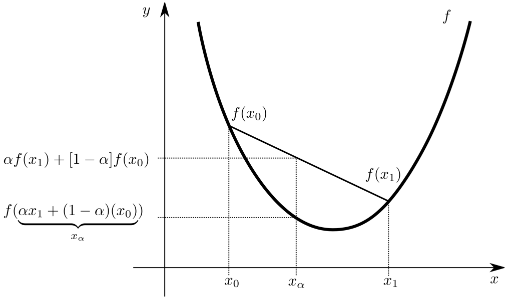


Definition

A function $f : \mathbb{R}^d \to \mathbb{R}$ is strictly convex if for all $(x, y) \in \mathbb{R}^d$ and all $\alpha \in]0, 1[$

$$f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha)f(y)$$

Interpretation : The graph of a cvx fct is always strictly below the segment joining 2 points on the graph.



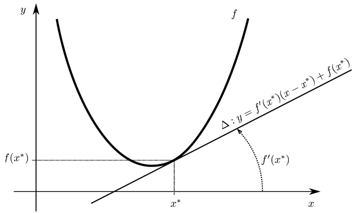
Convexity: characterisation

Convexity and linear approx

- If f is cvx and differentiable then for all x, y

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle$$

- Interpretation : a cvx fct is always above its tangent hyperplane
- For strict convexity, the inequality is strict when $y \neq x$



Convexity: characterisation

Convexity and monotone gradient

- f is cvx and differentiable $\iff$ for all x, y

$$\langle \nabla f(y) - \nabla f(x), y - x \rangle \geqslant 0$$

- In 1D : f cvx $\iff$ f' non-decreasing
- For strict convexity, the inequality is strict when $y \neq x$

Convexity and Hessian

- f is cvx and twice differentiable $\iff$ its Hessian is semi-definite positive for all x, i.e.

$$\nabla^2 f(w) \text{ or } H[f](x) \succcurlyeq 0$$

- In 1D : f cvx $\iff f'' \geqslant 0$
- For strict convexity, the Hessian is definite positive.

Back to the problem of minimizing

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

Note that f is convex iff

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

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

Classes of functions of interest

Definition

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

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

The gradient is L-Lipschitz continuous.

Another characterization of L-smooth

If f is twice differentiable, this is equivalent to assuming

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

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

Some L-smooth functions

- 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

$$L = \frac{1}{n} \lambda_{\max} \left(\sum_{i=1}^{n} x_i x_i^\top \right) = \frac{1}{n} \lambda_{\max} \left(X^T X \right)$$

with the (x_i)'s rows of X

Some L-smooth functions

- 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

$$L = \frac{1}{4n} \lambda_{\max} \left(\sum_{i=1}^{n} x_i x_i^\top \right) = \frac{1}{4n} \lambda_{\max} \left(X^T X \right)$$

with the (x_i)'s rows of X

Strong convexity

Def : Strong convexity

f is μ-strongly convex iff $\forall x, y$ and $\forall \lambda \in [0, 1]$

$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y) - \frac{\mu}{2}\lambda(1-\lambda)\|x-y\|_2^2$$

Strong convexity and linear approx.

A differentiable function f is μ-strongly convex if $\forall x, y$

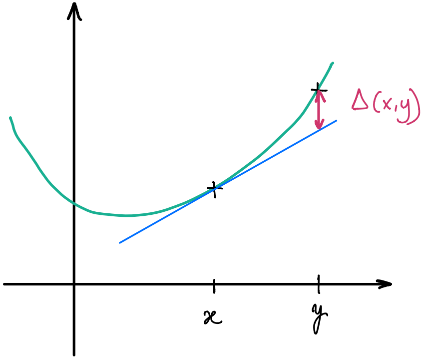
$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y-x\|_2^2$$

Strong convexity and Hessian

A twice-differentiable function f is μ-strongly convex if $\forall x$

$$\lambda_{\min}(\nabla^2 f(x)) \geqslant \mu$$

L-smooth and μ-strongly convex



- L-smooth : $\Delta(x, y) \leqslant \frac{L}{2}\|y - x\|_2^2$
- μ-strongly convex : $\Delta(x, y) \geqslant \frac{\mu}{2}\|y - x\|_2^2$

We define in this case

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

as the condition number of f.

Summary

A simple minimization problem

Now how to find

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

A key point is the following.

Lemma (The descent lemma)

If f is L-smooth, then

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

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

Proof of the descent lemma. 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

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

which proves the descent lemma. $\square$

Back to the minimization problem

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

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

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

Remark that

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

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

Which convergence guarantees?

Theorem (Convergence of gradient descent)

Assume that f has a minimizer $x^\star \in \mathbb{R}^d$ and that the gradient of f is Lipschitz continuous with Lipschitz constant $L > 0$:

$$\forall (x, y) \in (\mathbb{R}^d)^2 : \qquad \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

For a constant step size $\gamma_k = \frac{1}{L}$ ($\forall k \in \mathbb{N}$):

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

Which convergence guarantees?

Theorem (Convergence of gradient descent (strong convexity))

Assume that f has a minimizer $x^\star \in \mathbb{R}^d$, is μ-strongly convex ($\mu > 0$) and that the gradient of f is Lipschitz continuous with Lipschitz constant $L > 0$. For a constant step size $\gamma_k = \frac{1}{L}$ ($\forall k \in \mathbb{N}$):

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

and

$$\|x_k - x^\star\|^2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x_0 - x^\star\|_2^2,$$

Gradient descent : summary

Take-home message

- L-smooth : CV in $O(1/k)$ iterations - sublinear rate
- + μ-strong convexity :
  - CV in $O(c^k)$ $0 < c < 1$ - linear rate
  - CV for the iterates!

$\rightsquigarrow$ Complexity theory for first order methods

Can be accelerated

- Heavy ball methods
- Nesterov's acceleration

Let's not forget about g

Summary

Proximal Gradient Descent

Let's put back g:

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

and again

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

Some definitions

Definition (Proximal operator)

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

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

- If $g(w) = \lambda \|w\|_1$ then $\rightsquigarrow$ (soft-thresholding, cf TD)

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

- If $g(w) = \frac{\lambda}{2} \|w\|_2^2$ then $\rightsquigarrow$ (shrinkage)

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

Proximal gradient descent (PGD)

Algo : prox gradient descent (PGD)

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

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

- *Return* last w^k

Ex: Lasso

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

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

Theoretical guarantees for PGD

- 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

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

Comments

- Convergence rate is $O(1/k)$ (sublinear)
- ε-accuracy (namely $F(w^k) - F(w^*) \leqslant \varepsilon$) achieved after $O(L/\varepsilon)$ iterations
- Is it possible to improve the $O(1/k)$ rate? It's very slow!
- Again using: *strong convexity*

Recall

f is μ-*strongly convex* if

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

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

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

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

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

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

Theoretical guarantees for PGD

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

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

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

Comments

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

Summary

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

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

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

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

- *Return* last w^k

Accelerated PGD

Theorem

Accelerated proximal gradient descent needs

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

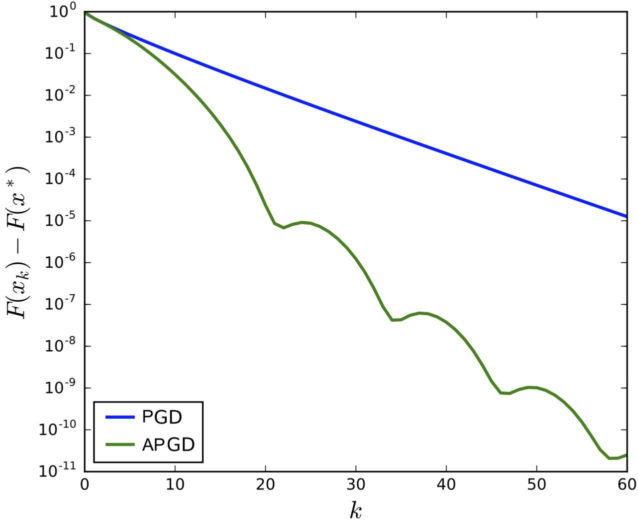
in the L-smooth case and

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

in the L-smooth and μ-strongly convex case

Remark You can also accelerate gradient descent with the same algorithm (by removing the prox)!

Accelerated PGD



Remark. APGD is *not* a descent algorithm, while PGD is

Summary

Newton method

Goal: still to minimize a function f (unconstrained problem)

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

- Gradient descent: 1st order method
  $\rightsquigarrow$ need for the gradient
- Newton method: 2nd order method
  $\rightsquigarrow$ need for the gradient and the Hessian

Newton iteration

$$w_{k+1} \leftarrow w_k - (\nabla^2 f(w_k))^{-1} \nabla f(w_k)$$

Proposition

If the Hessian is well-conditioned through the iterations, i.e.

$$\exists M > 0, \forall k \in \mathbb{N}, \|\nabla^2 f(w_k)\|_{2\to 2} \|(\nabla^2 f(w_k))^{-1}\|_{2\to 2} \leqslant M,$$

then, the Newton algorithm globally converges.

Convergence of the Newton method

Proposition

Assume that

- f *is convex,* C^2,
- *the Hessian is* M*-Lipschitz,*
- *the Hessian is locally lower-bounded, i.e.* $\exists \ell > 0$, *such that*

$$\nabla^2 f(w^\star) \succeq \ell \mathrm{Id}$$

- *the first iterate is not far from the solution* $w^\star$*:*

$$\|w_0 - w^\star\| < \bar{r} = \frac{2\ell}{3M}$$

Then the Newton method ensures that $\|w_k - w^\star\| \leqslant \bar{r}$ *for all* k *and it quadratically converges:*

$$\|w_{k+1} - w^\star\| \leqslant \frac{M\|w_k - w^*\|^2}{2(\ell - M\|w_k - w^*\|)}.$$

Summary

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)

A motivating proposition

Proposition

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

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

(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. $f(w + ze_j) \geqslant f(w)$ 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)

Algo : Exact coordinate descent (CD)

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

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

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

Ex: Least-squares linear regression

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

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

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

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

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

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

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

- Written like this, one update complexity is $n \times d$ (matrix-vector product $\mathrm{X}^{-j} w_{-j}$ 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* $r^t \leftarrow y - \mathrm{X} w^t$ we can write an update as

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

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

Theoretical guarantees for CD

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)

Algo : Coordinate gradient descent (CGD)

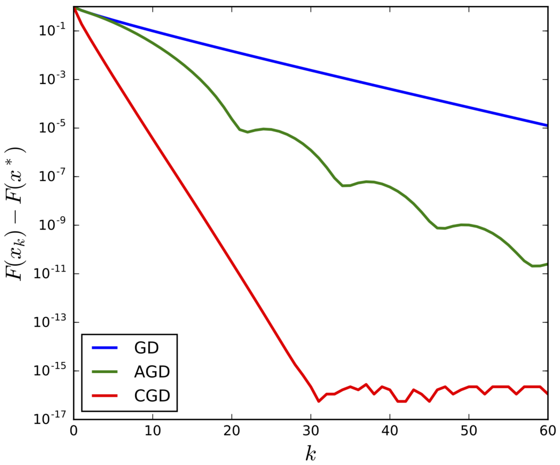
- For $k = 1, \ldots,$
- Choose $j \in \{1, \ldots, d\}$
- Compute

$$
\begin{aligned}
w_j^{k+1} &= w_j^k - \eta_j \nabla_{w_j} f(w^k) \\
w_{j'}^{k+1} &= w_{j'}^k \quad \text{for } j' \neq j
\end{aligned}
$$

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

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



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

The answer is...

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

$$\mathbb{E} f(w^{k+1}) - f(w^*) \leqslant \frac{d}{d+k}\left(\left(1 - \frac{1}{d}\right)(f(w^0) - f(w^*)) + \frac{1}{2}\|w^0 - w^*\|_L^2\right)$$

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 coordinates $j = (t \mod d) + 1$ the bound is much worse.

Comparison with gradient descent

- GD achieves ε-precision with

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

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

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

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

GD vs. CGD

- So, this is actually

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

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

$$L_j = \left\| X^j \right\|_2^2 = \| \mathrm{X} e_j \|_2^2 \leqslant \max_{u : \|u\|_2 = 1} \| \mathrm{X} u \|_2^2 = \lambda_{\max}(\mathrm{X}^\top \mathrm{X}) = L$$

- 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 $L_j = O(L/d)$

$\rightsquigarrow$This explains roughly why CGD is much faster than GD for ML problems

What's next?

- What about non-smooth penalization using CGD ?
- What if I want to use an ℓ^1 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)

Algo : Proximal coordinate gradient descent (PCGD)

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

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

where we recall that

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

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

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

Summary

Recall

We want to minimize

$$F(w) = f(w) + g(w)$$

- f is goodness-of-fit

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

- g is penalization, where main examples are

$$g(w) = \frac{\lambda}{2} \|w\|_2^2 \quad \text{(ridge)} \qquad g(w) = \lambda \|w\|_1 \quad \text{(lasso)}$$

Gradient descent

Gradient descent

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

- To achieve ε-precision, if f is L-smooth then the number of iterations is
$$O(L/\varepsilon),$$
- if f is also μ-strongly convex then the number of iterations is
$$O\left(\frac{L}{\mu}\log(1/\varepsilon)\right)$$

In terms of numerical cost, one should say

$$\rightsquigarrow O\left(n\frac{L}{\mu}\log(1/\varepsilon)\right)$$

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

Full gradient

These methods are said 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

Problem

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

A first estimator

Choosing uniformly at random $I \in \{1, \dots, n\}$, 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 $\nabla f(w)$

$\rightsquigarrow$ 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)

Algo : 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)

Comparison : numerical motivation

When f is μ-strongly con)vex and L-smooth (and if again the "unit" is complexity of $O(d)$)

- Full gradient descent

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

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

- Stochastic gradient descent

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

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

Take-home message

It does not depend on n for SGD !

Averaging against noise

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

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

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 for SGD

Theorem

If:

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

we have a convergence rate

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

and if moreover

- *f is μ-strongly convex*

the rate is

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

Both achieved by ASGD (average SGD)

Under strong convexity, GD versus SGD is

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

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)

Variance issue

- 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 $\mathbb{E}X = \nabla f(w)$

Problem

How to reduce $\mathbb{V}(X)$?

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

$$\mathbb{V}Z_\alpha = \alpha^2(\mathbb{V}X + \mathbb{V}C - 2\,\mathrm{Cov}(X, C))$$

- 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

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

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

SGD iterate with variance reduction

$$w^t \leftarrow w^{t-1} - \eta \left(\alpha(\nabla f_{i_t}(w^{t-1}) - \nabla f_{i_t}(\tilde{w})) + \nabla f(\tilde{w})\right)$$

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 (SAG)

Algo : 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 (SVRG)

Algo : Stochastic Variance Reduced Gradient (SVRG)

Input: starting point w^0, learning rate $\eta > 0$

Put $\tilde{w}_1 \leftarrow w^0$

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

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

$$w_k^{t+1} \leftarrow w_k^t - \eta \left(1 \cdot \left(\nabla f_i(w_k^t) - \nabla f_i(\tilde{w}_k)\right) + \nabla f(\tilde{w}_k)\right)$$

- Set

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

Return last w_k^t

SAGA

Algo : SAGA

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

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

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

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

Return last w^t

Prox-version

Stochastic Variance Reduced Gradient

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

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

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

SAGA

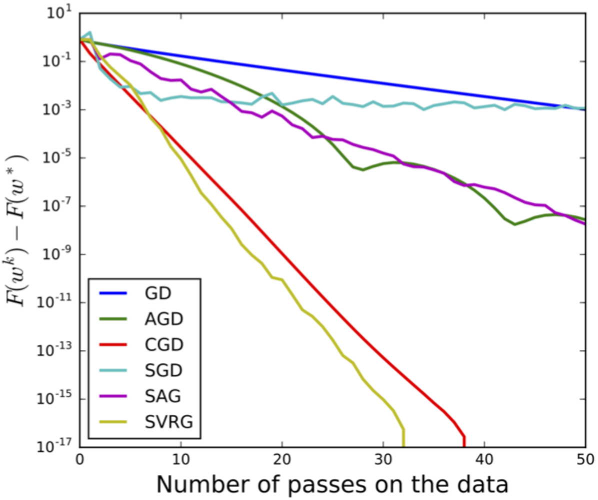
If $F = f + g$ with g prox-capable, use

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

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



Theoretical guarantees

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

Theorem (For SAG)

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

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

The rate is typically faster than gradient descent!

Theoretical guarantees

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

Theorem (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

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

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

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 (w/ unit in $O(d)$)

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

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

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

Some practical aspects

Memory

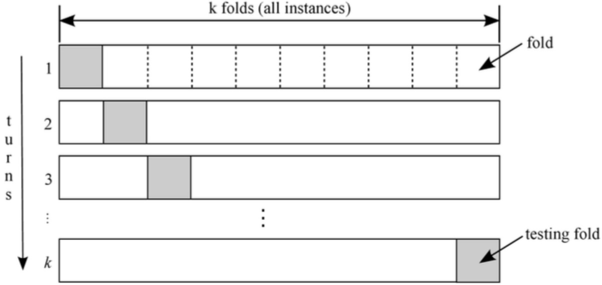
- SAG and SAGA requires extra memory: need to save all the previous gradients!
- Actually no...

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

  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 $v = 1, \ldots, V$



Question

How to do it with SGD type algorithms?

- 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

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

Algo :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