Optimization 2

Master Data Science

S. Gaïffas

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

1. Stochastic optimization
    - Stochastic Gradient Descent (SGD)
    - Beyond SGD
2. Supervised learning recipes (bis)
    - File indexing
    - Lazy-updates
    - Cross-validation
    - Warm starting

1 Stochastic optimization
- Stochastic Gradient Descent (SGD)
- Beyond SGD

2 Supervised learning recipes (bis)
- File indexing
- Lazy-updates
- Cross-validation
- Warm starting

Quick recap on convex optimization

We want to minimize

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$$

where $f_i(\theta) = \ell(y_i, \langle x_i, \theta\rangle) + \frac{\lambda}{2}\|\theta\|_2^2$

- Ridge Regression
- Ridge Logistic Regression
- etc.

Quick recap on convex optimization

Full (Batch) Gradient Descent

$$\theta^k \leftarrow \theta^{k-1} - \eta_k \nabla f(\theta^{k-1})$$

You've seen:

- If f convex and L-smooth then for $\eta_k = 1/L$

$$f(\theta^k) - f(\theta_*) \leq \frac{2L\|\theta^0 - \theta_*\|}{k+1} \quad (1)$$

  where $\theta_* \in \operatorname{argmin}_\theta f(\theta)$

- Acceleration (Nesterov, Fista): rate improvement $O(1/k^2)$
- Linesearch

If f is also μ-strongly convex, then linear convergence

$$f(\theta^k) - f(\theta_*) \leq \left(1 - \frac{L}{\mu}\right)^k (f(\theta_0) - f(\theta_*)) \quad (2)$$

Stochastic optimization

What if n (and d) is large?

- Each iteration of a full gradient method has complexity $O(nd)$
- I can't put $n \times d$ floats (32 or 64 bits) in my memory

Size of big data makes a modern computer look old: go back to "old" algorithms

- Idea: in machine learning, objective functions are averages of losses

Stochastic Gradient Descent

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

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

- $\nabla f_I(\theta)$ is an *unbiased* but very noisy estimate of the full gradient $\nabla f(\theta)$
- Computation of $\nabla f_I(\theta)$ only requires the I-th line of data ($O(d)$ and smaller for sparse data, see next)

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) algorithm (Robbins and Monro 1951)

- At each iteration, load a line of data chosen randomly (requires an index for fast random access on the hard drive)
- Compute gradient for this line of data
- Do a descent step, using this gradient, and repeat

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD)

- **Input**: starting point θ^0, sequence of learning rates $\{\eta_t\}_{t \geq 0}$
- For $t = 1, 2, \ldots$ until *convergence* do
  - Pick at random (uniformly) i in $\{1, \ldots, n\}$
  - Put
$$\theta^t = \theta^{t-1} - \eta_t \nabla f_i(\theta^{t-1})$$
- **Return** last θ^t

- 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)

Stochastic Gradient Descent

- Note that if i is chosen uniformly at random in $\{1, \dots, n\}$

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

  where $\mathcal{F}_t =$ information until iteration t (relative to random sampling of indexes)
- Namely, SGD uses very noisy unbiased estimations of the full gradient
- Learning rate η_t usually chosen as $\eta_t \approx Ct^{-\alpha}$ with $\alpha \in [1/2, 1]$.
- Linesearch: η_t is a valid learning rate if

$$f_i(\theta^t - \eta_t \nabla f_i(\theta^t)) \leq f_i(\theta^t) - \frac{\eta_t}{2} \|\nabla f_i(\theta^t)\|_2^2$$

Stochastic Gradient Descent

Polyak-Ruppert averaging (ASGD)

- Use SGD iterates $\{\theta^t\}$ but return $\bar{\theta}^t = \frac{1}{t}\sum_{t'=1}^{t} \theta^{t'}$
- Computed "online"

$$\bar{\theta}^t \leftarrow \frac{1}{t}\theta^{t-1} + \frac{t-1}{t}\bar{\theta}^{t-1} \tag{3}$$

- Leads to better results, cf:

  `http://leon.bottou.org/projects/sgd`

Stochastic Gradient Descent

Theoretical knowledge on SGD. Typical assumptions

- Each f_i is L-smooth (gradient L-lipshitz)
- f is μ-strongly convex

- Non strongly convex: rate $O(1/\sqrt{t})$ for ASGD with $\eta_t = O(1/\sqrt{t})$
- μ-Strongly convex: rate $O(1/(\mu t))$ for ASGD with $\eta_t = O(1/(\mu t))$

Both SGD and ASGD are **slow**. Best case is $O(1/t)$ convergence when f is strongly convex, while $O(e^{-\rho t})$ for FG

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)

Stochastic vs Batch optimization

- Gradient descent:

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

  $O(nd)$ iteration but linear convergence $O(e^{-\rho t})$ (strongly cvx case)

- Stochastic gradient descent:

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

  $O(d)$ iteration but slow convergence $O(1/t)$ (strongly cvx case)

Do a fast algorithm with $O(d)$ iteration exist?

Variance reduction

- Put $X = \nabla f_I(\theta)$ with I uniformly chosen at random in $\{1, \ldots, n\}$
- We want to use Monte Carlo samples to approximate $\mathbb{E}X = \nabla f(\theta)$
- We find out C s.t. $\mathbb{E}C$ is easy to compute and such that C 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

Idea: combine SGD with variance reduction

$$\theta^t \leftarrow \theta^{t-1} - \eta\Big(\alpha\big(\nabla f_{i_t}(\theta^{t-1}) - \nabla f_{i_t}(\varphi^{t-1})\big) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\varphi^{t-1})\Big)$$

where $\nabla f_i(\varphi^{t-1})$ is the "last computed" gradient of ∇f_i along the iterations

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

Variance reduction with SAG

Stochastic Average Gradient (SAG, Bach et al. 2013)

- **Input**: starting point θ_0, learning rate $\eta > 0$
- For $t = 1, 2, \ldots$ until *convergence* do
  - Pick at random (uniformly) i_t in $\{1, \ldots, n\}$
  - Put
$$g_t(i) = \begin{cases} \nabla f_i(\theta^{t-1}) & \text{if } i = i_t \\ g_{t-1}(i) & \text{otherwise} \end{cases}$$
    and compute
$$\theta^t = \theta^{t-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_t(i)$$
- **Return** last θ^t

Variance reduction with SAG

Assume

- Each f_i is L-smooth
- f is μ-strongly convex
- $\eta_t = 1/(16L)$ constant
- Initialize using one epoch of SGD

Non-strongly convex case:

$$\mathbb{E}[f(\theta^t) - f(\theta_*)] \leq O(\frac{\sqrt{n}}{t})$$

Strongly convex case:

$$\mathbb{E}[f(\theta^t) - f(\theta_*)] \leq O\Big(\frac{1}{n\mu} + \frac{L}{n}\Big) \exp\Big(-t\Big(\frac{1}{8n} \wedge \frac{\mu}{16L}\Big)\Big)$$

Improves a lot FG and SGD algorithms

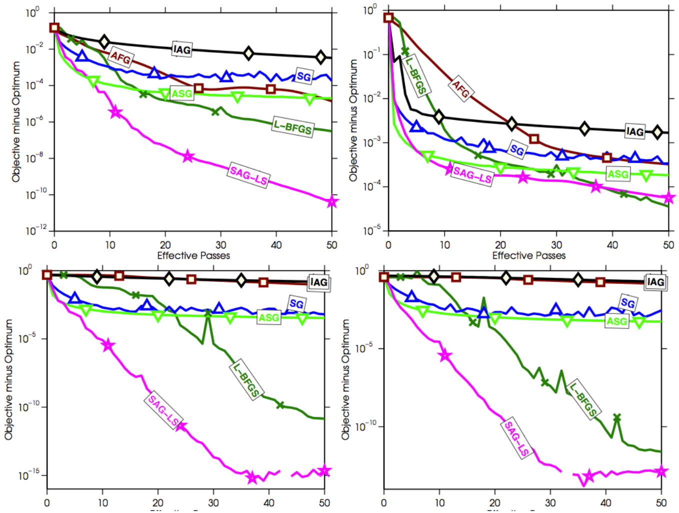
Variance reduction

- Complexity $O(d)$ instead of $O(nd)$ at each iteration
- Choice of a **fixed** step-size $\eta > 0$ possible
- But extra memory required: need to save all the previous gradients.

  Hopefully
  $$\nabla f_i(\theta) = \ell'(y_i, \langle x_i, \theta \rangle) x_i,$$
  so only need to save $\ell(y_i, \langle x_i, \theta \rangle)$. Memory footprint is $O(n)$ instead of $O(nd)$. If $n = 10^7$, this is 76 Mo

Variance reduction with SAG

Comparison of convergence [Le Roux el al. 2012]



Some practical remarks: file indexing

Now some practical problems / tricks around this

- Need to **index large data files** to be able to read lines at random fast

  Many tools to do this. In `Hadoop`[1] there is the `ArrayFile.Reader` that does that. It's only (roughly) 3x slower than a sequential read of the file.

  Stochastic optimization algorithm also work when using random shuffling of lines at beginning of each epoch: allows to further improve I/O.

[1] http://hadoop.apache.org

Some practical remarks: lazy updates

Feature vectors are usually very sparse (words counts). Complexity of the iteration of a stochastic optimization algorithm can reduced from $O(d)$ to $O(s)$, where s is the sparsity of the features. Important since $d \approx 10^6$ while $s \approx 10^3$

For minimizing

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

an iteration of SGD writes

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

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

Some practical remarks: lazy updates

Trick: put $\theta^t = s_t \beta^t$, with $s_t \in [0,1]$ and $s_t = (1 - \eta_t \lambda) s_{t-1}$

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

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 $O(s)$.

Some practical remarks: lazy updates

- Just check that s_t is not too small once in a while, in this case put $s_t = 1$ and update θ_t and β_t
- Write the algorithm using only β_t, return $s_t \beta_t$ in the end

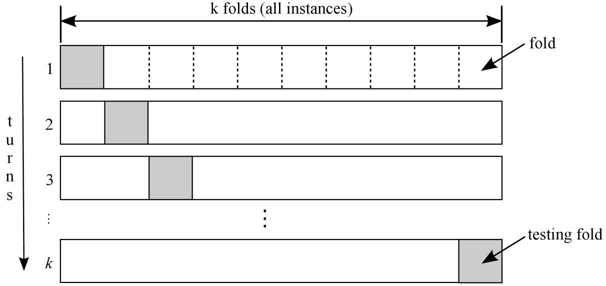
Now, complexity of one iteration of a stochastic algorithm is $O(s)$, while an approach based on FG methods is $O(nd)$ without using sparsity

Some practical remarks: cross-validation

Choice of penalization parameter λ by V-fold with SGD

Quick recap on V-fold:

- 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 $v = 1, \ldots, V$



Some practical remarks: cross-validation

- I don't load the full data in memory
- I can't use V-Fold this way when I'm using an SGD-based solver

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 $\hat{\theta}_{v'}$ using line i
- Update the testing error of $\hat{\theta}_v$ using line i

Cross-validation: warm starting

- So I have many optimization problems to solve for choosing λ!
- If I'm using V-Fold cross-validation, and a choose a set $\Lambda = \{\lambda_1, \ldots, \lambda_M\}$ of values for Λ, it is $V \times |\Lambda|$ problems
- But solutions $\hat{\theta}_{\lambda_{j-1}}$ and $\hat{\theta}_{\lambda_j}$ are going to be close when λ_{j-1} and λ_j are

Use **warm starts**:

- Fix parameters $\lambda_1 < \lambda_2 < \cdots < \lambda_M$
- Put $\theta_0 = 0$ (I don't know where to start)
- For $m = M, \ldots, 1$
  - Put $\lambda = \lambda_m$
  - Solve the problems starting at θ_0 for this value of λ (on each fold)
  - Keep the solutions $\hat{\theta}$ (test it, save it...)
  - Put $\theta_0 \leftarrow \hat{\theta}$

This allows to solve much more rapidly the sequence of problems