Machine Learning 2 – MAP569

Stéphane Gaïffas

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

This afternoon: **evaluated** practical session

- `jupyter notebook` **only**
- Must be done by **pairs of students** (we'll die correcting your work otherwise, and it's nice to have friends)
- **All** students must send their work, using the **moodle platform**
- Work must be sent **before sunday 12th december at 23:55**
- This means that **students in a pair send the same file**
- The jupyter notebook filename must contain both students names lowercase, with a `_and_` separation symbol. For instance:

  `bonnie_parker_and_clyde_barrow.ipynb`

No evaluation if you don't respect this EXACTLY

Questions must be asked on slack as much as possible:

- you'll get answers quicker
- You'll get answers **today** only

Sign-in using the following link with your `@polytechnique` mail

`https://map569.slack.com/signup`

Supervised learning setting

- features $x_i \in \mathbb{R}^d$
- labels $y_i \in \{-1, 1\}$ (binary classification), or
- labels $y_i \in \mathbb{R}$ (regression)

Loss function examples

- least-squares loss $\ell(y, y') = \frac{1}{2}(y - y')^2$ (linear regression)
- logistic loss $\ell(y, y') = \log(1 + e^{-yy'})$ (logistic regression)

We want to minimize

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

where f is goodness-of-fit

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

and g is penalization, where main ewamples are

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

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 $i_1, \dots, i_t$, denoted $\mathcal{F}_t$

If i_t is chosen uniformly at random in $\{1, \dots, n\}$ 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 $\operatorname{var} X$?

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 $t = 1, 2, \dots$ until *convergence* do

- Pick uniformly at random i_t in $\{1, \dots, n\}$
- 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 $w_k^0 \leftarrow \widetilde{w}_1$
    • 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
      $$w^t \leftarrow w^{t-1} - \eta\Big(\nabla f_{i_t}(w^{t-1}) - g_{t-1}(i_t) + \frac{1}{n}\sum_{i=1}^{n} g_{t-1}(i)\Big)$$
    • 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

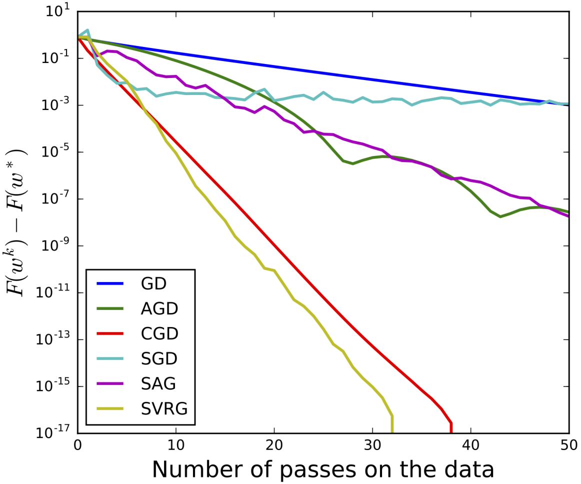
If $F = f + g$ with g prox-capable, use

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

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



You will code all of it this afternoon

Theoretical guarantees

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

For SAG

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

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

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

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

[we will prove that later...]

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

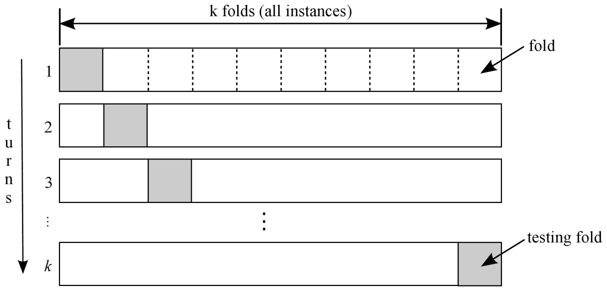
$$n\frac{L}{\mu}\log(1/\varepsilon) \gg \left(n + \frac{L_{\max}}{\mu}\right)\log(1/\varepsilon)$$

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 $\lambda = \lambda_1, \dots, \lambda_M$, 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 $m = M, \dots, 1$

- 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

[Convergence proofs for SGD on the blackboard]

Thank you!

(and have nice holidays)