OPTIMIZATION FOR MACHINE LEARNING

Regularized, large-scale and distributed optimization

November 20, 2023

Today: Coordinate descent methods

Two lectures remaining: Nov 23 Constrained optimization (V. Duval)
Dec 4 Distributed optimization (C. Royer)

Exam: December 14 (open book)

Course project: Deadline January 19, 2024

COORDINATE DESCENT METHODS

① Basics

Context: $\underset{x \in \mathbb{R}^d}{\text{minimize}} \; f(x)$, $f : \mathbb{R}^d \to \mathbb{R}$ C^1

$\boxed{d \gg 1}$

Recall GD

$$\underbrace{x_{k+1}}_{\in \mathbb{R}^d} = \underbrace{x_k}_{\in \mathbb{R}^d} - \alpha_k \underbrace{\nabla f(x_k)}_{\in \mathbb{R}^d} \qquad \alpha_k > 0$$

At every iteration, GD:

- Computes a d-dimensional vector ($\nabla f(x_k)$)
- Updates a d-dimensional vector (x_k)

Basic coordinate descent iteration

$$x_{k+1} = x_k - \alpha_k \nabla_{j_k} f(x_k) \, e_{j_k} \qquad \alpha_k > 0, \quad j_k \in \{1, \dots, d\}$$

$\nabla_{j_k} f(x_k)$: j_kth coordinate of $\nabla f(x_k)$

$$= \frac{\partial f}{\partial x_{j_k}}(x_k) \in \mathbb{R}$$

$$e_{j_k} = \begin{array}{c} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow j_k \\ \\ \\ \\ \end{matrix} \end{array} \in \mathbb{R}^d$$

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_d}(x) \end{bmatrix}$$

The iteration reduces to a scalar update

$$[x_{k+1}]_{j_k} = [x_k]_{j_k} - \alpha_k \nabla_{j_k} f(x_k) \qquad j_k \in \{1, \dots, d\}$$

$\Rightarrow$ Only update 1 coordinate of the iterate at a time

$\Rightarrow$ $\nabla_{j_k} f(x_k)$ might depend on all coordinates of x_k in general, but when the function is partially separable the cost of evaluating $\nabla_{j_k} f(x_k)$ can be much lower than the cost of evaluating $\nabla f(x_k)$

Def: . A function $f: \mathbb{R}^d \to \mathbb{R}$ is called <u>separable</u> if

$$\forall x \in \mathbb{R}^d, \quad f(x) = \sum_{j=1}^{d} f_j(x_j), \quad \text{where } f_j : \mathbb{R} \to \mathbb{R}$$

and $f_j(x_j)$ only depends on x_j

Ex) $f(x) = \|x\|_1 = \sum_{j=1}^{d} |x_j|$

$f(x) = \frac{1}{2}\|x\|_2^2 = \sum_{j=1}^{d} \left(\frac{1}{2} x_j^2\right)$

. A function $f: \mathbb{R}^d \to \mathbb{R}$ is called <u>partially separable</u>

if $\quad f(x) = \sum_{g \in G} f_g(x_g)$, where $f_g : \mathbb{R}^{|g|} \to \mathbb{R}$

$g \subseteq \{1, \dots, d\}$

$\bigcup_{g \in G} = \{1, \dots, d\}$ and every f_g depends on

optional but common

a subset of the coordinates of x $\left(x_g = [x_j]_{j \in g}\right)$

Ex) $f(x) = \sum_{g \in G} \|x_g\|_2$ is partially separable

Classical strategies for choosing j_k

a) Cyclic coordinate descent   Cycle through $\{1, -, d\}$ in that order

$$j_0 = 1,\ j_1 = 2, -, j_{d-1} = d,\ j_d = 1, -, j_{2d-1} = d, j_{2d} = 1, \ldots$$

↳ After d iterations, all coordinates of the iterate have been updated

b) Randomized cyclic coordinate descent

↳ Every d iterations, choose a random permutation of $\{1, -, d\} \longrightarrow \{\sigma(1), -, \sigma(d)\}$

↳ Choose the indices for the next d iterations as $\sigma(1), \ldots, \sigma(d)$

$\Rightarrow$ For separable functions, d iterations of $\left|\begin{array}{l}\text{Cyclic CD} \\ \text{Randomized cyclic CD}\end{array}\right|$ are equivalent to 1 iteration of GD

c) Randomized CD: j_k chosen at random in $\{1, -, d\}$

↳ The randomized techniques have better theoretical guarantees than cyclic CD!

Connection between stochastic gradient and coordinate descent

- First viewpoint

$$\nabla f(x_k) = \sum_{j=1}^{d} \nabla_j f(x_k) e_j = \frac{1}{d} \sum_{j=1}^{d} \underbrace{\left(d \nabla_j f(x_k)\right) e_j}$$

$$(*) \quad \boxed{\nabla f(x_k) = \frac{1}{d} \sum_{j=1}^{d} \nabla f_j (x_k)}$$

gradient is a finite sum

where we define

$\nabla f_j(x_k)$ to be $(d \nabla_j f(x_j)) e_j$

Recall: For SG, we had $\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i (x_k)$

$\longrightarrow$ Can view randomized CD as a special case of SG applied to $(*)$

- Second viewpoint

Consider a finite-sum problem

$$(\mathcal{P}) \quad \underset{x \in \mathbb{R}^d}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^{n} l_i (a_i^T x) + \lambda \Omega(x)$$

where $\quad a_i \in \mathbb{R}^d$ data vector

$l_i : \mathbb{R} \to \mathbb{R}$ loss function that may depend on the i-th data point

$\Omega : \mathbb{R}^d \to \mathbb{R}$ regularization term, $\lambda > 0$

l_i, Ω convex

The Fenchel dual (see V. Duval's lectures) of (P) is

$$(D) \quad \underset{y \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{n} \sum_{j=1}^{n} \ell_j^*(-[y]_j) + \lambda \Omega^*\left(\frac{1}{\lambda n} A^T y\right)$$

separable

where $A^T = [a_1 \cdots a_n] \in \mathbb{R}^{d \times n}$ $(A \in \mathbb{R}^{n \times d})$

and $\forall$ convex function $\phi : \mathbb{R}^d \to \mathbb{R}$, the convex conjugate of ϕ is

$$\phi^* : \mathbb{R}^d \to \mathbb{R}$$
$$y \mapsto \sup_{z \in \mathbb{R}^d} \left(z^T y - \phi(z) \right)$$

- We can apply SG to (P) using

$$\nabla \left[\ell_{i_k}(a_{i_k}^T x_k) \right] = \nabla \ell_{i_k}(a_{i_k}^T x_k)\, a_{i_k}$$

$\Rightarrow$ Produces a sequence $\{x_k\}$ $\quad i_k \in \{1, \dots, n\}$

- We can also apply CD to (D) $\Rightarrow$ Produces a sequence $\{y_k\}$ using

$$\nabla \left[\ell_{j_k}^*(-[y]_{j_k}) \right] = \nabla_{j_k} \left(\frac{1}{n} \sum_{j=1}^{n} \ell_j^*(-[y]_j) \right)$$

$j_k \in \{1, \dots, n\}$

With the same sequence of random indices $\left(\{i_k\} \text{ for SG}, \{j_k\} \text{ for CD}, i_k = j_k \ \forall k \right)$,

Then the two methods are equivalent

and x_k is equivalent to $\frac{1}{\lambda n} A^T y_k$

equal up to a constant factor

→ Useful in overparameterized settings where $d \gg n$

Remark: Randomized coordinate descent is sometimes called stochastic dual descent because of this connection

Block coordinate descent (~ Batch SG)

$$x_{k+1} = x_k - \alpha_k \sum_{j \in B_k} \nabla_j f(x_k) e_j \qquad \alpha_k > 0$$

where $B_k \subseteq \{1, \dots, d\}$ is a block of coordinates

⚠ Difference with SG: B_k never contains duplicates of indices

→ Randomized block CD: draw indices without replacement

$|B_k| = 1 \Rightarrow$ Randomized CD

$|B_k| = d \Rightarrow$ GD

Proximal coordinate descent

↳ Applies to $\underset{x \in \mathbb{R}^d}{\text{minimize}} \; f(x) + \lambda \Omega(x)$

↳ Particularly interesting when Ω is separable

$$\Omega(x) = \sum_{j=1}^{d} \Omega_i(x_i)$$

$\Rightarrow$ The iteration of a proximal CD then becomes

The objective function of the subproblem is separable in x

$$x_{k+1} \in \operatorname*{argmin}_{x \in \mathbb{R}^d} \left\{ f(x_k) + \left[\nabla_{j_k} f(x_k) e_{j_k}\right]^T (x - x_k) + \frac{1}{2\alpha_k} \|x - x_k\|^2 + \lambda \Omega_{j_k}([x]_{j_k}) \right\}$$

replace $\nabla f(x_k)$ by a coordinate vector $\nabla_{j_k} f(x_k) e_{j_k}$

regularization w.r.t. j_kth coordinate

$\Rightarrow$ This can be rewritten as a 1-dimensional problem

$$[x_{k+1}]_{j_k} \in \operatorname*{argmin}_{c \in \mathbb{R}} \left\{ f(x_k) + \nabla_{j_k} f(x_k)\,(c - [x_k]_{j_k}) + \frac{1}{2\alpha_k}(c - [x_k]_{j_k})^2 + \lambda \Omega_{j_k}(c) \right\}$$

(The other coordinates of x_{k+1} are identical to that of x_k)

$\Rightarrow$ cheap updates, easy to extend to block CD

$\Rightarrow$ CD methods are used in sparse optimization because many sparsity-inducing regularizers are (partially) separable

2) Analysis of Coordinate descent

Focus on the basic variant

$$x_{k+1} = x_k - \alpha_k \nabla_{j_k} f(x_k) e_{j_k}$$

$j_k \in \{1, \dots, d\}$

Theorem (Powell, 1973): Cyclic CD doesn't work!

↳ Powell gave a counterexample

$d=3$

$$f\left(x=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}\right) = -\left(x_1x_2+x_1x_3+x_2x_3\right) + \sum_{i=1}^{3}\max\left(|x_i|-1,0\right)$$

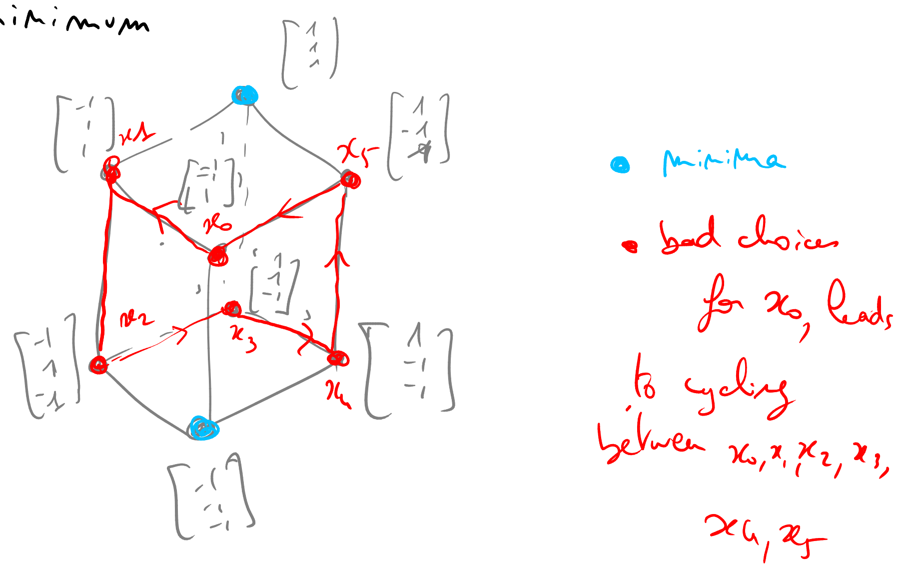
f C^1, $\underset{x\in\mathbb{R}^3}{\operatorname{argmin}} f(x) = \left\{\begin{bmatrix}1\\1\\1\end{bmatrix},\begin{bmatrix}-1\\-1\\-1\end{bmatrix}\right\}$

If α_k is chosen through exact minimization

(i.e. $\alpha_k = \underset{\alpha>0}{\operatorname{argmin}} f\left(x_k - \alpha \nabla_{j_k} f(x_k)e_{j_k}\right)$)

and start from $x_0=\begin{bmatrix}1\\-1\\1\end{bmatrix}$, then the method cycles between 6 points, none of which is a minimum

$x_0 \to x_1$ $i=2$
$x_1 \to x_2$ $i=3$
$x_2 \to x_3$ $i=1$
$x_3 \to x_4$ $i=2$
$x_4 \to x_5$ $i=3$
$x_5 \to x_0$ $i=1$



↳ Very pathological example: choosing another x_0 (other than the red vertices) leads to convergence
↳ But explains why CD methods were not investigated

much until late 2000s where they became of interest in ML

→ ML point of view: • Apply CD on problems that do not fall into the bad cases identified by Powell (and others)

• Focus on randomized CD (because of its ties to SG)

Some theoretical results

We consider $\underset{x \in \mathbb{R}^d}{\text{minimize}}\ f(x)$, where f is $C^{1,1}_L$

∇f L-Lipschitz

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

f is μ-strongly convex

∇f is coordinate-wise Lipschitz continuous, i.e.

$\forall j = 1..d,\ \forall (x,y) \in (\mathbb{R}^d)^2$

$$|\nabla_j f(x) - \nabla_j f(y)| \leq L_j \|x-y\|$$

$L_j > 0$

If $L_{max} = \max_{1 \leq j \leq d} L_j$, we have $\boxed{1 \leq \frac{L}{L_{max}} \leq d}$

$\varphi : \mathbb{R}^d \to \mathbb{R}^m$

L-Lip.

$$L \approx \max_{\{(x_i, y_i)\}} \frac{\|\varphi(x_i) - \varphi(y_i)\|}{\|x_i - y_i\|}$$

Th) Consider K iterations of randomized CD with

- j_k drawn uniformly at random in $\{1, \dots, d\}$ $\forall k$
- $\alpha_k = \frac{1}{L_{j_k}}$ $\forall k$

Then

$$\mathbb{E}_{j_0, \dots, j_{K-1}}\left[f(x_K)\right] - \min_{x \in \mathbb{R}^d} f(x) \le \underbrace{\left(1 - \frac{\mu}{d L_{max}}\right)^K}_{\to 0 \text{ as } K \to \infty} \left(f(x_0) - \min_{x \in \mathbb{R}^d} f(x)\right)$$

Convergence rate in expected value (randomized CD!)

$1 - \frac{\mu}{d L_{max}} \in [0, 1)$

For GD, would get

$1 - \frac{\mu}{L} \le 1 - \frac{\mu}{d L_{max}}$

$\Rightarrow$ Better rate in the worst case for GD

$d L_{max}$: "price" for using a single coordinate at every iterate

⊖ Worse than GD, expected value

⊕ Better rate for large d in terms of updates of coordinates of x_k, still get convergence to the optimum (unlike SG)

<u>NB</u>: CD decreases the objective at every iteration (unlike SG)

Th) Consider K iterations of cyclic CD under the same assumptions, then

$$f(x_K) - \min_{x \in \mathbb{R}^d} f(x) \le \left(1 - \frac{\mu}{2L_{max}\left(1 + d\frac{L^2}{L_{max}}\right)}\right)^{K/d} ,$$

$$\times \left(f(x_0) - \min_{x \in \mathbb{R}^d} f(x)\right)$$

⊖ Rate of CV is worse than GD

$$\left(1-\frac{\mu}{L}\right)^k \quad \text{vs} \quad \left(1-\frac{\mu}{\ldots}\right)^{k/d}$$

⇒ Even in terms of coordinate updates, the rate will not be better than that of gradient descent

↳ These results can be improved on specific problems which is usually how CD methods are analyzed in ML

3) CD for parallel/distributed optimization

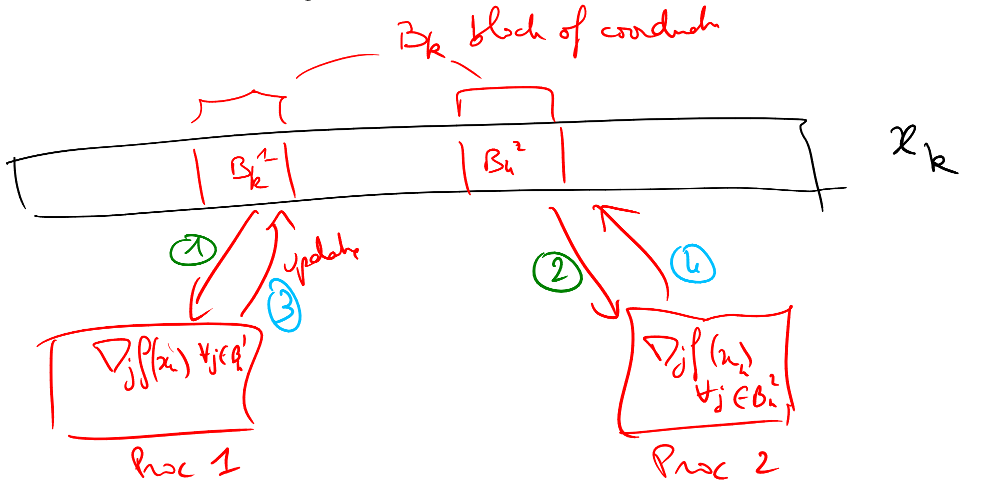
Setup: $x_k \in \mathbb{R}^d$ is too large to be updated via GD
⇒ stored in memory

Iteration k of CD

→ Distribute steps along (blocks of) coordinates to different processors

→ Every processor updates their coordinates

→ Synchronization after every update



1, 2 } run in parallel

3, 4 } run in parallel

→ Synchronization: wait until ③ and ④ are completed before sending the coordinates again

⟹ Because of synchronization, this process does not bring a lot of benefit

⟹ still efficient when the problem is separable (synchronization is not an issue)

Asynchronous CD : Do not wait for the other processors!

- Initialization: $x_0 \in \mathbb{R}^d$ stored, shared iteration count $k=0$
- Repeat loop (for all processors)

  choose $j_k \in \{1, \dots, d\}$

  $$[x_{k+1}]_{j_k} \leftarrow [x_k]_{j_k} - \alpha_k \nabla_{j_k} f(\hat{x}_k)$$

  $\hat{x}_k$: value of x_k when $\nabla_{j_k} f(x_k)$ was computed

  $\alpha_k > 0$

  Asynchronous update

  $k \leftarrow k+1$

  synchronization only for k

↳ Surprisingly, this can work!

⟹ For convex f, if

- every coordinate of x_k is updated infinitely often
- every coordinate of $\hat{x}_k$ is updated infinitely often

then $x_k \xrightarrow[k \to \infty]{} x^* \in \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} f(x)$

↳ Follow-up of asynchronous CD: Asynchronous SG!

⇒ Hogwild! : Asynchronous SG with theory and good practical success

⇒ NeurIPS 2011, won the test-of-time award in 2020