Coordinate Update Methods in Image Processing and Machine Learning

Wotao Yin (UCLA Math)

Zhimin Peng & Tianyu Wu (UCLA), Yangyang Xu (IMA), Ming Yan (MSU)

SIAM Imagin — May 23rd, 2016

Goal and Approach

- Goal: Develop fast and scalable algorithms for more complicated optimization problems
- Approach: Coordinate update, which is fast and becomes even faster when running in a parallel fashion

ERM and stochastic methods

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \; r(x) + \frac{1}{N} \sum_{i=1}^{N} f_i(x)$$

- interested in large N
- often called *empirical risk minimization* (ERM)
- nice structures: f_i's are smooth and r is proximable
- stochastic methods: SG, SAG, SAGA, SVRG, Finito
- issues: update $x \in \mathbb{R}^m$; model is restricted

Coordinate descent methods

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \; f(x_1, \ldots, x_m) + \frac{1}{m} \sum_{i=1}^{m} r_i(x_i)$$

- interested in large m
- nice structures: f is smooth and r_i's are separably proximable
- coordinate (descent) methods: (shuffled) cyclic, random, greedy, parallel
- issues: do not work with total variations and linear constraints

Issues with coupled nonsmooth functions

$$\underset{x=(x_1,x_2)}{\text{minimize}}\, f(x_1,x_2) + r(x_1,x_2)$$

- joint minimization condition

$$0 = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}, \quad \text{where } \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \nabla f(x_1,x_2), \quad \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \in \partial r(x_1,x_2)$$

- coordinate minimization conditions:

$$\min_{x_1} : \quad 0 = p_1 + q_1 \quad \text{where } p_1 = \nabla_1 f(x_1,x_2),\; q_1 \in \partial_1 r(x_1,x_2)$$

$$\min_{x_2} : \quad 0 = p_2 + q_2 \quad \text{where } p_2 = \nabla_1 f(x_1,x_2),\; q_2 \in \partial_2 r(x_1,x_2)$$

- The issue:

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \nabla f(x_1,x_2) \checkmark \qquad\qquad \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \notin \partial r(x_1,x_2) \text{ ✗}$$

(but true if r is separable)

Today:

Consider

- $r(x)$ is nonsmooth and coupled, and
- linear constraints $Ax = b$.

Approach:

- minimization $\Rightarrow$ primal-dual optimality condition
- then, apply forward-backward splitting (recover Chambolle-Pock)
- then, apply sequential coordinate update
- then, apply asynchronous parallel coordinate update

Expectation: 3–5x faster with 1 core; $>$16x faster with 32 cores; much larger problems can be solved

Primal dual optimality condition

$$\underset{x \in \mathcal{H}}{\text{minimize}} \;\; f(x) + g(y) + h(Ax)$$

- example: $\text{minimize}_u \;\; \frac{1}{2}\|Bu - b\|^2 + \iota_{[0,255]}(u) + \lambda\|\nabla u\|_1.$
- optimality condition: $0 \in (\nabla f + \partial g + A^T \circ \partial h \circ A)(x).$
- operator form:

$$0 \in \Bigg(\underbrace{\begin{bmatrix} \nabla f & 0 \\ 0 & 0 \end{bmatrix}}_{\text{operator } \mathcal{A}} + \underbrace{\begin{bmatrix} \partial g \\ \partial h^* \end{bmatrix} + \begin{bmatrix} 0 & A^\top \\ -A & 0 \end{bmatrix}}_{\text{operator } \mathcal{B}} \Bigg) \underbrace{\begin{bmatrix} x \\ s \end{bmatrix}}_{z},$$

- let U be invertible, $\gamma > 0$; note that $\mathcal{A}$ and $(U + \gamma\mathcal{B})$ is single-valued

$$\begin{aligned} 0 \in (\mathcal{A} + \mathcal{B})z &\Leftrightarrow -\gamma\mathcal{A}z \in \gamma\mathcal{B}z \\ &\Leftrightarrow (U - \gamma\mathcal{A})z \in (U + \gamma\mathcal{B})z \\ &\Leftrightarrow (U + \gamma\mathcal{B})^{-1}(U - \gamma\mathcal{A})z = z \end{aligned}$$

- the forward-backward splitting algorithm

$$z^{k+1} = (U + \gamma\mathcal{B})^{-1}(U - \gamma\mathcal{A})z^k$$

(converges if γ is sufficiently small; diverges unboundedly if no fixed-point)

- set a proper U to cancel terms, yielding Chambolle-Pock (Condat-Vu):

$$\begin{cases} s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma A x^k), \\ x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T(2s^{k+1} - s^k))), \end{cases}$$

Coordinate update

$$z^{k+1} = Tz^k$$

- suppose $z \in \mathbb{R}^m$; write $T = (T_1, \ldots, T_m)$ so that $T_i z = (Tz)_i$
- coordinate update: pick a coordinate $i \in [m]$

$$z_i^{k+1} = T_i z^k$$

  keep $z_j^{k+1} = z_j^k \ \forall j \neq i$.
- benefits: small memory footprint, parallelizable or sequential
- requirement: $\mathrm{cost}[T_i z] \ \sim \ \frac{1}{m}\mathrm{cost}[Tz]$

Coordinate friendly operator

- allow: maintain quantities $\mathcal{M}(z)$ in memory and update them
- let: z^+ be obtained from z after the update $T_i z$
- **Definition**: T is *coordinate friendly* if

$$\mathrm{cost}\Big[\{z, \mathcal{M}(z)\} \mapsto \{z^+, \mathcal{M}(z^+)\}\Big] = O\Big(\frac{1}{m}\mathrm{cost}\big[z \mapsto Tz\big]\Big)$$

- generalizes to coordinate blocks in obvious ways

Chambolle-Pock is coordinate friendly

$$\begin{cases} s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma A x^k), \\ x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T(2s^{k+1} - s^k))), \end{cases}$$

Theorem

Assumptions: *Functions g and h are separable and proximable. ∇f is coordinate friendly.*

Conclusion: *The Chambolle-Pock algorithm is coordinate friendly.*

Also applies to other primal-dual, e.g., Chen-Huang-Zhang, Inverse Problems'13

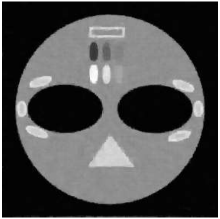
See **UCLA CAM 16-13** for

- coordinate-friendly ∇f
- coordinate-friendly operator splitting: forward-backward, backward-forward, Douglas-Rachford, ADMM ...
- applications in machine learning, SVM, (group) LASSO, logistic regression, SOCP, TV imaging, portfolio optimization, etc.
- parallel update s and x (instead of updating s then x)
- overlapped block coordinate updates (to save computation)

CT simulation

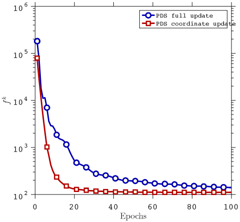
- 284×284, 90 beam projections, 362 measurements each beam
- partitioned to 284 columns (blocks), run 100 epochs

(a) Phantom image



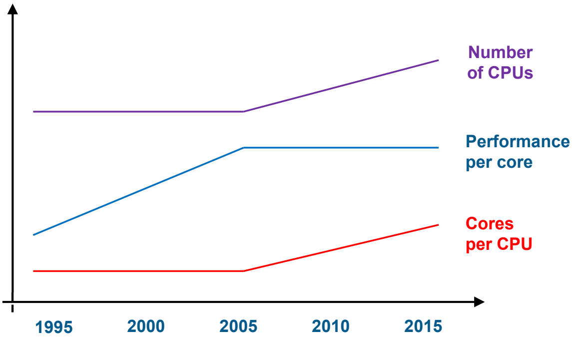
(b) Recovered by PDS

(c) Recovered by PDS coord



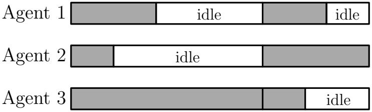
(d) Objective function value

35 Years of CPU Trend



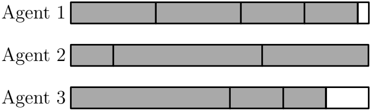
D. Henty. Emerging Architectures and Programming Models for Parallel Computing, 2012.

Sync-parallel versus async-parallel



Synchronous

(wait for the slowest)



Asynchronous

(non-stop, no wait)

ARock[1]: Async-parallel coordinate update

- $x = (x_1, \ldots, x_m) \in \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$
- p agents, possibly $\neq m$
- $S_i = I - T_i$
- each agent randomly picks $i \in \{1, \ldots, m\}$:

$$\begin{aligned} x_i^{k+1} &\leftarrow x_i^k - \eta_k S_i(x^{k-d_k}) \qquad\qquad \text{(only update } x_i\text{)} \\ x_{\neq i}^{k+1} &\leftarrow x_{\neq i}^k \end{aligned}$$

- $0 \le d_k \le \tau$, maximum delay

[1] Peng-Xu-Yan-Y.'15

Convergence guarantees for ARock (async-parallel random coordinate descent)

notation:

- m is # coordinates
- τ is the maximum delay
- uniform selection $p_i \equiv \frac{1}{m}$

Theorem (almost sure convergence)

Assume that T is nonexpansive and has a fixed point. Use step sizes $\eta_k \in [\epsilon, \frac{1}{2m^{-1/2}\tau+1})$, $\forall k$. Then, with probability one, $x^k \rightharpoonup x^* \in \mathrm{Fix}T$.

Consequence:

- $O(1)$ step size if $\tau \sim \sqrt{m}$
- assuming similar agents, linear speedup with up to $O(\sqrt{m})$ parallel agents.
- **do not bother with synchronizing until** $p > O(\sqrt{m})$

Example: sparse logistic regression

- ℓ_1 regularized logistic regression:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ \lambda \|x\|_1 + \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + \exp(-b_i \cdot a_i^T x)\right), \tag{1}$$

- n features, N labeled samples
- each sample $a_i \in \mathbb{R}^n$ has its label $b_i \in \{1, -1\}$

Name	N (#samples)	n (#features)	#nonzeros in $\{a_1, \ldots, a_N\}$
rcv1	20,242	47,236	1,498,952
news20	19,996	1,355,191	9,097,916

Speedup tests

- implemented in C++ and OpenMP.
- 32 cores shared memory machine.

#cores	rcv1				news20			
	Time (s)		Speedup		Time (s)		Speedup	
	async	sync	async	sync	async	sync	async	sync
1	122.0	122.0	1.0	1.0	591.1	591.3	1.0	1.0
2	63.4	104.1	1.9	1.2	304.2	590.1	1.9	1.0
4	32.7	83.7	3.7	1.5	150.4	557.0	3.9	1.1
8	16.8	63.4	7.3	1.9	78.3	525.1	7.5	1.1
16	9.1	45.4	13.5	2.7	41.6	493.2	14.2	1.2
32	4.9	30.3	**24.6**	**4.0**	22.6	455.2	**26.1**	**1.3**

Conclusions:

- Many problems in imaging and machine learning are *coordinate friendly*
- Coordinate update is faster
- Coordinate update can be (asynchronously) parallelized

References: UCLA CAM ??-?? and CAM 16-13

Also: ARock talk by Ming Yan (tomorrow 2pm, MS37 in Alvarado Ballroom G)