Incremental gradients, parallel methods
(Optml++ Meeting 5)

Suvrit Sra

Massachusetts Institute of Technology

OPTML++, Fall 2015

Outline

- Lect 1: Recap on convexity
- Lect 1: Recap on duality, optimality
- Lect 2: First-order optimization algorithms
- Lect 3: Operator splitting
- Lect 4: Stochastic and incremental methods
- Lect 5: Parallel / sparse-data methods

Recap

Stochastic gradient

Method	Assumptions	Full	Stochastic
Subgradient	convex	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$
Subgradient	strongly cvx	$O(1/k)$	$O(1/k)$

So using stochastic subgradient, solve n times faster.

Method	Assumptions	Full	Stochastic
Gradient	convex	$O(1/k)$	$O(1/\sqrt{k})$
Gradient	strongly cvx	$O((1-\mu/L)^k)$	$O(1/k)$

– For smooth problems, stochastic gradient needs more iterations
– Widely used in ML, rapid initial convergence
– Several speedup techniques studied, but worst case remains same

Incremental Gradient Methods

Method	Assumptions	Rate
Gradient	convex	$O(1/k)$
Gradient	strongly cvx	$O((1-\mu/L)^k)$
Stochastic	strongly cvx	$O(1/k)$
SAG	strongly convex	$O((1-\min\{\frac{\mu}{n}, \frac{1}{8n}\})^k)$

This speedup also observed in practice

Complicated convergence analysis

Similar rates for many other methods

- stochastic dual coordinate (SDCA); [Shalev-Shwartz, Zhang, 2013]
- stochastic variance reduced gradient (SVRG); [Johnson, Zhang, 2013]
- proximal SVRG [Xiao, Zhang, 2014]
- hybrid of SAG and SVRG, SAGA (also proximal); [Defazio et al, 2014]
- accelerated versions [Lin, Mairal, Harchoui; 2015]
- asynchronous hybrid SVRG [Reddi et al. 2015]
- incremental Newton method, S2SGD and MS2GD, ...

Incremental gradient methods

$$\min \quad F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

Incremental gradient methods

$$\min \quad F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- Incremental gradient methods

$$x_{k+1} = x_k - \frac{\eta_k}{n} \nabla f_{i(k)}(x_k), \quad k \geq 0.$$

Incremental gradient methods

$$\min \quad F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- Incremental gradient methods

$$x_{k+1} = x_k - \frac{\eta_k}{n} \nabla f_{i(k)}(x_k), \quad k \geq 0.$$

- View as gradient-descent with perturbed gradients

$$x_{k+1} = x_k - \frac{\eta_k}{n} (\nabla F(x_k) + e_k)$$

Incremental gradient methods

$$\min \quad F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- Incremental gradient methods

$$x_{k+1} = x_k - \frac{\eta_k}{n} \nabla f_{i(k)}(x_k), \quad k \geq 0.$$

- View as gradient-descent with perturbed gradients

$$x_{k+1} = x_k - \frac{\eta_k}{n} (\nabla F(x_k) + e_k)$$

- Perturbation slows down rate of convergence. Typically $\eta_k = O(1/k)$; convergence rate also $O(1/k)$ (sublinear).

Incremental gradient methods

$$\min \quad F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- Incremental gradient methods

$$x_{k+1} = x_k - \frac{\eta_k}{n} \nabla f_{i(k)}(x_k), \quad k \geq 0.$$

- View as gradient-descent with perturbed gradients

$$x_{k+1} = x_k - \frac{\eta_k}{n} (\nabla F(x_k) + e_k)$$

- Perturbation slows down rate of convergence. Typically $\eta_k = O(1/k)$; convergence rate also $O(1/k)$ (sublinear).
- Can we reduce impact of perturbation to speed up?

Incremental Gradient Methods

$$\min F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The incremental gradient method (IGM)

- Let $x_0 \in \mathbb{R}^n$
- For $k \geq 0$

Incremental Gradient Methods

$$\min F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The incremental gradient method (IGM)

- Let $x_0 \in \mathbb{R}^n$
- For $k \geq 0$
  1. Pick $i(k) \in \{1, 2, \ldots, n\}$ uniformly at random
  2. $x_{k+1} = x_k - \eta_k \nabla f_{i(k)}(x_k)$

Incremental Gradient Methods

$$\min F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The incremental gradient method (IGM)

- Let $x_0 \in \mathbb{R}^n$
- For $k \geq 0$
  1. Pick $i(k) \in \{1, 2, \ldots, n\}$ uniformly at random
  2. $x_{k+1} = x_k - \eta_k \nabla f_{i(k)}(x_k)$

$g \equiv \nabla f_{i(k)}$ may be viewed as a **stochastic gradient**

Incremental Gradient Methods

$$\min F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The incremental gradient method (IGM)

- Let $x_0 \in \mathbb{R}^n$
- For $k \geq 0$
  1. Pick $i(k) \in \{1, 2, \ldots, n\}$ uniformly at random
  2. $x_{k+1} = x_k - \eta_k \nabla f_{i(k)}(x_k)$

$g \equiv \nabla f_{i(k)}$ may be viewed as a **stochastic gradient**

$g := g^{\text{true}} + e$, where e is mean-zero noise: $\mathbb{E}[e] = 0$

Incremental Gradient Methods

- Index $i(k)$ chosen uniformly from $\{1, \ldots, n\}$
- Thus, **in expectation:**

$$\mathbb{E}[g] =$$

Incremental Gradient Methods

- Index $i(k)$ chosen uniformly from $\{1, \ldots, n\}$
- Thus, **in expectation:**

$$\mathbb{E}[g] = \mathbb{E}_i[\nabla f_i(x)]$$

Incremental Gradient Methods

- Index $i(k)$ chosen uniformly from $\{1, \ldots, n\}$
- Thus, **in expectation:**

$$\mathbb{E}[g] = \mathbb{E}_i[\nabla f_i(x)] = \sum_i \tfrac{1}{n} \nabla f_i(x) =$$

Incremental Gradient Methods

- Index $i(k)$ chosen uniformly from $\{1, \ldots, n\}$
- Thus, **in expectation:**

$$\mathbb{E}[g] = \mathbb{E}_i[\nabla f_i(x)] = \sum_i \tfrac{1}{n} \nabla f_i(x) = \nabla F(x)$$

- Alternatively, $\mathbb{E}[g - g^{\text{true}}] = \mathbb{E}[e] = 0$.
- We call g an **unbiased estimate** of the gradient
- Here, we **obtained** g in a two step process:
  - **Sample:** pick an index $i(k)$ unif. at random
  - **Oracle:** Compute a random gradient based on $i(k)$

Incremental Gradient Methods

- Index $i(k)$ chosen uniformly from $\{1, \ldots, n\}$
- Thus, **in expectation:**

$$\mathbb{E}[g] = \mathbb{E}_i[\nabla f_i(x)] = \sum_i \tfrac{1}{n} \nabla f_i(x) = \nabla F(x)$$

- Alternatively, $\mathbb{E}[g - g^{\text{true}}] = \mathbb{E}[e] = 0$.
- We call g an **unbiased estimate** of the gradient
- Here, we **obtained** g in a two step process:
  - **Sample:** pick an index $i(k)$ unif. at random
  - **Oracle:** Compute a random gradient based on $i(k)$
- Individual g_k values can vary a lot
- Variance ($\mathbb{E}[\|g - g^{\text{true}}\|^2]$) influences convergence rate

Controlling variance

- Instead of using $g_k = \nabla f_{i(k)}(x_k)$, **correct** it by using **true gradient** every $m \geq n$ steps (recall: $F = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$)

Controlling variance

- Instead of using $g_k = \nabla f_{i(k)}(x_k)$, **correct** it by using **true gradient** every $m \geq n$ steps (recall: $F = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$)
- Reduces variance of $g_k(x_k, \xi_k)$; speeds up convergence

Controlling variance

- Instead of using $g_k = \nabla f_{i(k)}(x_k)$, **correct** it by using **true gradient** every $m \geq n$ steps (recall: $F = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$)
- Reduces variance of $g_k(x_k, \xi_k)$; speeds up convergence

$$\begin{aligned} \nabla F(\bar{x}) &= \tfrac{1}{m}\sum\nolimits_i f_i(\bar{x}) \\ x_{k+1} &= x_k - \eta_k \underbrace{[\nabla f_{i(k)}(x_k) - \nabla f_{i(k)}(\bar{x}) + \nabla F(\bar{x})]}_{g_k(x_k,\xi_k)} \end{aligned}$$

Controlling variance

- Instead of using $g_k = \nabla f_{i(k)}(x_k)$, **correct** it by using **true gradient** every $m \geq n$ steps (recall: $F = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$)
- Reduces variance of $g_k(x_k, \xi_k)$; speeds up convergence

$$\begin{aligned} \nabla F(\bar{x}) &= \tfrac{1}{m}\sum\nolimits_i f_i(\bar{x}) \\ x_{k+1} &= x_k - \eta_k \underbrace{[\nabla f_{i(k)}(x_k) - \nabla f_{i(k)}(\bar{x}) + \nabla F(\bar{x})]}_{g_k(x_k,\xi_k)} \end{aligned}$$

- Thus, with $\xi_k = i(k)$, $\mathbb{E}_\xi[g_k|x_k] = \nabla F(x_k)$
  But with lower variance!

Controlling variance

- Instead of using $g_k = \nabla f_{i(k)}(x_k)$, **correct** it by using **true gradient** every $m \geq n$ steps (recall: $F = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$)
- Reduces variance of $g_k(x_k, \xi_k)$; speeds up convergence

$$\begin{aligned} \nabla F(\bar{x}) &= \tfrac{1}{m}\sum\nolimits_i f_i(\bar{x}) \\ x_{k+1} &= x_k - \eta_k \underbrace{[\nabla f_{i(k)}(x_k) - \nabla f_{i(k)}(\bar{x}) + \nabla F(\bar{x})]}_{g_k(x_k,\xi_k)} \end{aligned}$$

- Thus, with $\xi_k = i(k)$, $\mathbb{E}_\xi[g_k|x_k] = \nabla F(x_k)$
  But with lower variance!

Say $\bar{x}, x_k \to x^*$. Then $\nabla F(\bar{x}) \to 0$. Thus, if $\nabla f_i(\bar{x}) \to \nabla f_i(x^*)$, then

$$\nabla f_i(x_k) - \nabla f_i(\bar{x}) + \nabla F(\bar{x}) \to \nabla f_i(x_k) - \nabla f_i(x^*) \to 0.$$

SVRG

- For $s \geq 1$:
  1. $\bar{x} \leftarrow \bar{x}_{s-1}$
  2. $\bar{g} \leftarrow \nabla F(\bar{x})$ (full gradient computation)
  3. $x_0 = \bar{x}; \quad t \leftarrow \text{RAND}(1, m)$ (randomized stopping)
  4. For $k = 0, 1, \ldots, t-1$
     - Randomly pick $i(k) \in [1..m]$
     - $x_{k+1} = x_k - \eta_k(\nabla f_{i(k)}(x_k) - \nabla f_{i(k)}(\bar{x}) + \bar{g})$
  5. $\bar{x}_s \leftarrow x_t$

SVRG

- For $s \geq 1$:
  1. $\bar{x} \leftarrow \bar{x}_{s-1}$
  2. $\bar{g} \leftarrow \nabla F(\bar{x})$ (full gradient computation)
  3. $x_0 = \bar{x}; \quad t \leftarrow \text{RAND}(1, m)$ (randomized stopping)
  4. For $k = 0, 1, \ldots, t-1$
     - Randomly pick $i(k) \in [1..m]$
     - $x_{k+1} = x_k - \eta_k(\nabla f_{i(k)}(x_k) - \nabla f_{i(k)}(\bar{x}) + \bar{g})$
  5. $\bar{x}_s \leftarrow x_t$

Theorem Assume each $f_i(x)$ is smooth, and $F(x)$ strongly-convex. Then, for sufficiently large n, there is $\alpha < 1$ s.t.

$$\mathbb{E}[F(\bar{x}_s) - F(x^*)] \leq \alpha^s [F(\bar{x}_0) - F(x^*)]$$

Coordinate descent

Block Coordinate Descent

$$\min f(x) = f(x_1, \ldots, x_K), \text{ where } x_i \in \mathbb{R}^{n_i}$$

Block Coordinate Descent

$$\min f(x) = f(x_1, \ldots, x_K), \text{ where } x_i \in \mathbb{R}^{n_i}$$

Assumption: Gradient of block i is Lipschitz continuous

$$\|\nabla_i f(x + E_i h) - \nabla_i f(x)\| \leq L_i \|h\|$$

Block gradient $\nabla_i f(x)$ is projection of full grad: $E_i^T \nabla f(x)$

Block Coordinate Descent

$$\min f(x) = f(x_1, \ldots, x_K), \text{ where } x_i \in \mathbb{R}^{n_i}$$

Assumption: Gradient of block i is Lipschitz continuous

$$\|\nabla_i f(x + E_i h) - \nabla_i f(x)\| \leq L_i \|h\|$$

Block gradient $\nabla_i f(x)$ is projection of full grad: $E_i^T \nabla f(x)$

Block Coordinate "Gradient" Descent

- Using lemma: $f(y) \leq f(x) + \langle \nabla f(x),\, y - x \rangle + \frac{L}{2}\|y - x\|^2$, we get

$$f(x + E_i h) \leq f(x) + \langle \nabla_i f(x),\, h \rangle + \frac{L_i}{2}\|h\|^2, \quad \text{for } i = 1, \ldots, n.$$

Block Coordinate Descent

$$\min f(x) = f(x_1, \ldots, x_K), \text{ where } x_i \in \mathbb{R}^{n_i}$$

Assumption: Gradient of block i is Lipschitz continuous

$$\|\nabla_i f(x + E_i h) - \nabla_i f(x)\| \leq L_i \|h\|$$

Block gradient $\nabla_i f(x)$ is projection of full grad: $E_i^T \nabla f(x)$

Block Coordinate "Gradient" Descent

- Using lemma: $f(y) \leq f(x) + \langle \nabla f(x),\, y - x \rangle + \frac{L}{2}\|y - x\|^2$, we get

$$f(x + E_i h) \leq f(x) + \langle \nabla_i f(x),\, h \rangle + \frac{L_i}{2}\|h\|^2, \quad \text{for } i = 1, \ldots, n.$$

- BCD algorithm:
  1. repeatedly go through blocks in "some" order
  2. minimize these upper bounds

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(x_k)$$

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(x_k)$$

- Update the impacted coordinates of x, formally

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(x_k)$$

- Update the impacted coordinates of x, formally

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} + h$$

$$x_{k+1} \leftarrow x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)$$

Randomized BCD

- For $k \geq 0$ (no init. of x necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(x_k)$$

- Update the impacted coordinates of x, formally

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} + h$$

$$x_{k+1} \leftarrow x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)$$

Notice: Original BCD had: $x_k^{(i)} = \operatorname{argmin}_h f(\ldots, \underbrace{h}_{\text{block } i}, \ldots)$

We'll call this BCM (**Block Coordinate Minimization**)

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \frac{L_i}{2} \|h\|^2$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned} x_{k+1} &= x_k + E_i h \\ f(x_{k+1}) &\leq f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned} x_{k+1} &= x_k + E_i h \\ f(x_{k+1}) &\leq f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \\ x_{k+1} &= x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k) \end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned}
x_{k+1} &= x_k + E_i h \\
f(x_{k+1}) &\leq f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \\
x_{k+1} &= x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k) \\
f(x_{k+1}) &\leq f(x_k) - \tfrac{1}{L_i}\|\nabla_i f(x_k)\|^2 + \tfrac{L_i}{2}\left\| -\tfrac{1}{L_i}\nabla_i f(x_k)\right\|^2
\end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned}
x_{k+1} &= x_k + E_i h \\
f(x_{k+1}) &\leq f(x_k) + \langle \nabla_i f(x_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \\
x_{k+1} &= x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k) \\
f(x_{k+1}) &\leq f(x_k) - \tfrac{1}{L_i}\|\nabla_i f(x_k)\|^2 + \tfrac{L_i}{2}\left\| -\tfrac{1}{L_i}\nabla_i f(x_k)\right\|^2 \\
f(x_{k+1}) &\leq f(x_k) - \tfrac{1}{2L_i}\|\nabla_i f(x_k)\|^2.
\end{aligned}$$

$$f(x_k) - f(x_{k+1}) \geq \tfrac{1}{2L_i}\|\nabla_i f(x_k)\|^2$$

Randomized BCD – analysis

Expected descent:

$$f(x_k) - \mathbb{E}[f(x_{k+1}|x_k)] \quad = \quad \sum_{i=1}^{d} p_i \big(f(x_k) - f(x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)) \big)$$

Randomized BCD – analysis

Expected descent:

$$
\begin{aligned}
f(x_k) - \mathbb{E}[f(x_{k+1}|x_k)] &= \sum_{i=1}^{d} p_i \big(f(x_k) - f(x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)) \big) \\
&\geq \sum_{i=1}^{d} \tfrac{p_i}{2L_i} \|\nabla_i f(x_k)\|^2
\end{aligned}
$$

Randomized BCD – analysis

Expected descent:

$$
\begin{aligned}
f(x_k) - \mathbb{E}[f(x_{k+1}|x_k)] &= \sum_{i=1}^{d} p_i \big(f(x_k) - f(x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)) \big) \\
&\geq \sum_{i=1}^{d} \tfrac{p_i}{2L_i} \|\nabla_i f(x_k)\|^2 \\
&= \tfrac{1}{2} \|\nabla f(x_k)\|_W^2 \quad \text{(suitable } W\text{)}.
\end{aligned}
$$

Randomized BCD – analysis

Expected descent:

$$\begin{aligned} f(x_k) - \mathbb{E}[f(x_{k+1}|x_k)] &= \sum_{i=1}^{d} p_i \big(f(x_k) - f(x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)) \big) \\ &\geq \sum_{i=1}^{d} \tfrac{p_i}{2L_i} \|\nabla_i f(x_k)\|^2 \\ &= \tfrac{1}{2} \|\nabla f(x_k)\|_W^2 \quad \text{(suitable } W\text{)}. \end{aligned}$$

What is the expected descent with uniform probabilities?

Randomized BCD – analysis

Expected descent:

$$\begin{aligned} f(x_k) - \mathbb{E}[f(x_{k+1}|x_k)] &= \sum_{i=1}^{d} p_i \big(f(x_k) - f(x_k - \tfrac{1}{L_i} E_i \nabla_i f(x_k)) \big) \\ &\geq \sum_{i=1}^{d} \tfrac{p_i}{2L_i} \|\nabla_i f(x_k)\|^2 \\ &= \tfrac{1}{2} \|\nabla f(x_k)\|_W^2 \quad \text{(suitable } W\text{)}. \end{aligned}$$

What is the expected descent with uniform probabilities?

Descent + more notation + some work yields

$$O(\tfrac{d}{\epsilon} \textstyle\sum_i L_i \|x_0^{(i)} - x_*^{(i)}\|^2)$$

as the iteration complexity of obtaining $\mathbb{E}[f(x_k)] - f^* \leq \epsilon$

Exercise

- Recall Lasso problem: $\min \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$
- Here $x \in \mathbb{R}^d$; use d blocks
- Show what the Randomized BCD iterations look like
- Recall 1D prox operations for $\lambda|\cdot|$ arise
- Try to implement it as efficiently as you can (do not copy or update vectors / coordinates unless necessary)

Exercise – pseudocode

Assuming d blocks, each update is scalar valued.

- Let $x_0 = 0$; $y_0 = Ax_0 - b = -b$
- For $k \geq 0$
  - Pick random coordinate $j \in [d]$
  - Compute $\alpha \leftarrow \langle a_j, y \rangle$ – i.e., $\nabla_j f(x_k)$
  - Min $\alpha h + \frac{L_j}{2} h^2 + \lambda |h|$

$$h = \text{prox}_{\lambda|\cdot|}(x_j - \tfrac{1}{L_j}\alpha)$$

$$h = \text{sgn}(x_j - \tfrac{1}{L_j}\alpha)\max(|x_j - \tfrac{1}{L_j}\alpha| - \lambda, 0)$$

  - Update: $x_{k+1} = x_k + he_j$
  - Update: $y_{k+1} \leftarrow y_k + ha_j$

Parallel Optimization

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden;

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden; node failure, network failure, load-balancing, etc.

Parallel computation – high level views

- Intuition: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden; node failure, network failure, load-balancing, etc.
- Synchronous vs. asynchronous computation

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$
- Problem is now over $\mathcal{H}^m := \mathcal{H} \times \mathcal{H} \times \cdots \times \mathcal{H}$ (m-times)

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$
- Problem is now over $\mathcal{H}^m := \mathcal{H} \times \mathcal{H} \times \cdots \times \mathcal{H}$ (m-times)
- **Consensus** constraint: $x_1 = x_2 = \ldots = x_m$

$$\min_{(x_1,\ldots,x_m)} \quad \sum_i f_i(x_i)$$
$$\text{s.t.} \quad x_1 = x_2 = \cdots = x_m.$$

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{1}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{1}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

- Can solve using proximal splitting methods (e.g., DR, ADMM)

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{1}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

- Can solve using proximal splitting methods (e.g., DR, ADMM)
- Each component of $f_i(x_i)$ independently in parallel
- Communicate / synchronize to ensure consensus
- Asynchronous versions exist (results from 2014, 2015)

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{1}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

- Can solve using proximal splitting methods (e.g., DR, ADMM)
- Each component of $f_i(x_i)$ independently in parallel
- Communicate / synchronize to ensure consensus
- Asynchronous versions exist (results from 2014, 2015)
- Alternatively, compute dual and apply $\|$ BCD

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_d)$$

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_d)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_d)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

- Can solve all d problems **independently** in **parallel**
- In theory: d times speedup possible compared to serial case

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_d)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

- Can solve all d problems **independently** in **parallel**
- In theory: d times speedup possible compared to serial case
- if objective "almost separable" we would still expect high speedup, governed by amount of **separability**
- Big data problems often have this "almost separable" structure!

Partial Separability

Consider the **sparse** data matrix

$$\begin{pmatrix} d_{11} & d_{12} & & \\ & d_{22} & d_{23} & \\ & & \ddots & \ddots \end{pmatrix} \in \mathbb{R}^{m \times n},$$

Partial Separability

Consider the **sparse** data matrix

$$\begin{pmatrix} d_{11} & d_{12} & & \\ & d_{22} & d_{23} & \\ & & \ddots & \ddots \end{pmatrix} \in \mathbb{R}^{m \times n},$$

- Objective $f(x) = \|Dx - b\|_2^2 = \sum_{i=1}^m (d_i^T x - b_i)^2$ also equals
$$(d_{11}x_1 + d_{12}x_2 - b_1)^2 + (d_{22}x_2 + d_{23}x_3 - b_2)^2 + \cdots$$
- Each term depends on only 2 coordinates
- Formally, we could write this as
$$f(x) = \sum\nolimits_{J \in \mathcal{J}} f_J(x),$$
where $\mathcal{J} = \{\{1,2\}, \{2,3\}, \cdots\}$
- Key point: $f_J(x)$ depends only on x_j for $j \in J$.

Partial Separability

Def. Let $\mathcal{J}$ be a collection of subsets of $\{1, \ldots, d\}$. We say f has **overlap degree** ω if it can be written as

$$f(x) = \sum_{J \in \mathcal{J}} f_J(x),$$

where each f_J depends only on x_j for $j \in J$, and

$$|J| \leq \omega \quad \forall J \in \mathcal{J}.$$

Example: If $D_{m \times n}$ is a sparse matrix, then $\omega = \max_{1 \leq i \leq m} \|d_i^T\|_0$

Partial Separability

Def. Let $\mathcal{J}$ be a collection of subsets of $\{1, \ldots, d\}$. We say f has **overlap degree** ω if it can be written as

$$f(x) = \sum_{J \in \mathcal{J}} f_J(x),$$

where each f_J depends only on x_j for $j \in J$, and

$$|J| \leq \omega \quad \forall J \in \mathcal{J}.$$

Example: If $D_{m \times n}$ is a sparse matrix, then $\omega = \max_{1 \leq i \leq m} \|d_i^T\|_0$

Exercise: Extend this notion to $x = (x^{(1)}, \ldots, x^{(K)})$

Hint: Now, f_J will depend only on $x^{(j)}$ for $j \in J$

Data sparse ML problems

$$\min \quad \sum_{i=1}^{n} \ell(\langle a_i, x \rangle) + \lambda \|x\|^2$$

Training data samples $a_1, \ldots, a_n$ are sparse

Data sparse ML problems

$$\min \quad \sum_{i=1}^{n} \ell(\langle a_i, x \rangle) + \lambda \|x\|^2$$

Training data samples $a_1, \ldots, a_n$ are sparse
Rewrite the above problem in the format

$$\min \quad \sum_{i=1}^{n} \Big(\ell(\langle a_i, x \rangle) + \lambda \sum_{u \in J_i} \frac{x_u^2}{d_u} \Big),$$

where J_i are the nonzero coordinates of a_i; d_u is the number of training samples nonzero in coordinate $u \in [d]$.

Data sparse ML problems

$$\min \quad \sum_{i=1}^{n} \ell(\langle a_i, x \rangle) + \lambda \|x\|^2$$

Training data samples $a_1, \ldots, a_n$ are sparse
Rewrite the above problem in the format

$$\min \quad \sum_{i=1}^{n} \Big(\ell(\langle a_i, x \rangle) + \lambda \sum_{u \in J_i} \frac{x_u^2}{d_u} \Big),$$

where J_i are the nonzero coordinates of a_i; d_u is the number of training samples nonzero in coordinate $u \in [d]$.

This is of the form (where $\mathcal{J} \subset 2^{[d]}$),

$$\min \quad \sum_{J \in \mathcal{J}} f_J(x_J)$$

Degree of overlap ω: maximum frequency any given feature

Parallel Stochastic Gradient

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, d\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

Parallel Stochastic Gradient

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, d\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)

Parallel Stochastic Gradient

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, d\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)
- Since the actual coordinate j can arise in various J, processors can overwrite each others' work.

Parallel Stochastic Gradient

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, d\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)
- Since the actual coordinate j can arise in various J, processors can overwrite each others' work.
- But if **partial overlaps**, coordinate j does not appear in too many different subsets J, method works!
- Several related approaches exist in the literature

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, d\}$

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, d\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(x_k)$$

$\longrightarrow$ w_i typically L_i; β depends on overlap ω

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, d\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(x_k)$$

$\longrightarrow$ w_i typically L_i; β depends on overlap ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, d\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(x_k)$$

$\longrightarrow$ w_i typically L_i; β depends on overlap ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees
- ♠ Theory requires **atomic updates**

Parallel BCD

1. Choose initial point $x_0 \in \mathbb{R}^d$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, d\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$x_{k+1}^{(i)} \leftarrow x_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(x_k)$$

$\longrightarrow$ w_i typically L_i; β depends on overlap ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees
- ♠ Theory requires **atomic updates**
- ♠ Implement **asynchronously** (use latest $x^{(i)}$ at each core)
- ♠ Theory of above method requires **guaranteed descent**
- ♠ Newer asynchronous CD methods also exist (see survey by Wright, 2015); e.g., methods that support **inconsistent reads**