Randomized Optimization Algorithms for Learning

Stephen Wright

University of Wisconsin-Madison

May 2013

Collaborators

UW-Madison colleagues and students:

Victor Bittorf
Ji Liu
Ben Recht
Chris Ré
Krishna Sridhar

Optimization, Learning, Data

Optimization is vital in data analysis:

- extract useful meaning from data,
- incorporate data into models,
- use big data to make good decisions.

Optimization is key to using data to inform models – and much of applied mathematics is about modeling (physical, engineered, economic systems).

Old (but still relevant) instances:

- Least-squares fitting,
- robust fitting (Huber, ℓ_1);
- data assimilation (to dynamic models);
- PDE-constrained optimization;
- model predictive control.

Data Analysis / Learning: Optimization Formulations

- A collection of data, from which we want to extract key information and/or make inferences about future / missing data.
- A parametrized model that captures the relationship between the data and the meaning we are trying to extract.
- An objective that measures quality of parameters: e.g. model / data discrepancies, deviation from prior knowledge or desired structure.

Other typical properties of learning problems are

- Big data;
- Need only low-medium accuracy;
- Have some prior knowledge about the model parameters;
- Have some desired structure for the parameters. Regularization.

In some cases, the optimization formulation is well settled: Least Squares, Robust Regression, Support Vector Machines, Logistic Regression, Recommender Systems.

In other areas, formulation is a matter of ongoing debate!

Topics

- Some Applications and Target Problems
- Stochastic Gradient
- Kaczmarz for $Ax = b$
- Stochastic Coordinate Descent

I. Canonical Application: Least-Squares Regression

Least Squares: Given a set of feature vectors $a_i \in \mathbb{R}^n$ and outcomes b_i, $i = 1, 2, \ldots, m$, find weights x on the features that predict the outcome accurately: $a_i^T x \approx b_i$.

Under certain assumptions on measurement error, can find a suitable x by solving a least squares problem

$$\min_x \frac{1}{2}\|Ax - b\|_2^2,$$

where the rows of A are a_i^T, $i = 1, 2, \ldots, m$.

Can modify for feature selection by adding a LASSO regularizer:

$$\textbf{LASSO:} \quad \min_x \frac{1}{2}\|Ax - b\|_2^2 + \tau\|x\|_1,$$

for some parameter $\tau > 0$. This identifies an approximate minimizer of the least-squares loss with few nonzeros (sparse).

Canonical Application: Support Vector Classification

Given many data vectors $x_i \in \mathbb{R}^n$, for $i = 1, 2, \ldots, m$, together with a label $y_i = \pm 1$ to indicate the class (one of two) to which x_i belongs.

Find z such that (usually) we have

$$x_i^T z \geq 1 \text{ when } y_i = +1 \text{ and } x_i^T z \leq -1 \text{ when } y_i = -1.$$

SVM with hinge loss:

$$f(z) = C \sum_{i=1}^{N} \max(1 - y_i(z^T x_i), 0) + \frac{1}{2}\|z\|^2,$$

where $C > 0$ is a parameter. **Dual formulation** is

$$\min_{\alpha} \frac{1}{2}\alpha^T K \alpha - \mathbf{1}^T \alpha \quad \text{subject to } 0 \leq \alpha \leq C\mathbf{1},\ y^T \alpha = 0,$$

where $K_{ij} = y_i y_j x_i^T x_j$.

- Subvectors of the gradient $K\alpha - \mathbf{1}$ can be computed cheaply.
- Nonlinear kernel: redefine $K_{ij} = y_i y_j k(x_i, x_j)$ for some positive semidefinite kernel function.

(Regularized) Logistic Regression

Seek odds function parametrized by $z \in \mathbb{R}^n$:

$$p_+(x; z) := (1 + e^{z^T x})^{-1}, \quad p_-(x; z) := 1 - p_+(z; w),$$

choosing z so that $p_+(x_i; z) \approx 1$ when $y_i = +1$ and $p_-(x_i; z) \approx 1$ when $y_i = -1$. Scaled, negative log likelihood function $\mathcal{L}(z)$ is

$$\mathcal{L}(z) = -\frac{1}{m}\left[\sum_{y_i=-1} \log p_-(x_i; z) + \sum_{y_i=1} \log p_+(x_i; z)\right]$$

Add regularizer $\tau\|z\|_1$ to select features.

M **classes**: $y_{ij} = 1$ if data point i is in class j; $y_{ij} = 0$ otherwise. $z_{[j]}$ is the subvector of z for class j.

$$f(z) = -\frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{M} y_{ij}(z_{[j]}^T x_i) - \log(\sum_{j=1}^{M} \exp(z_{[j]}^T x_i))\right] + \sum_{j=1}^{M} \|z_{[j]}\|_2^2.$$

Matrix Completion

Seek a matrix $X \in \mathbb{R}^{m \times n}$ with low rank that matches certain observations, possibly noisy.

$$\min_X \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2 + \tau\psi(X),$$

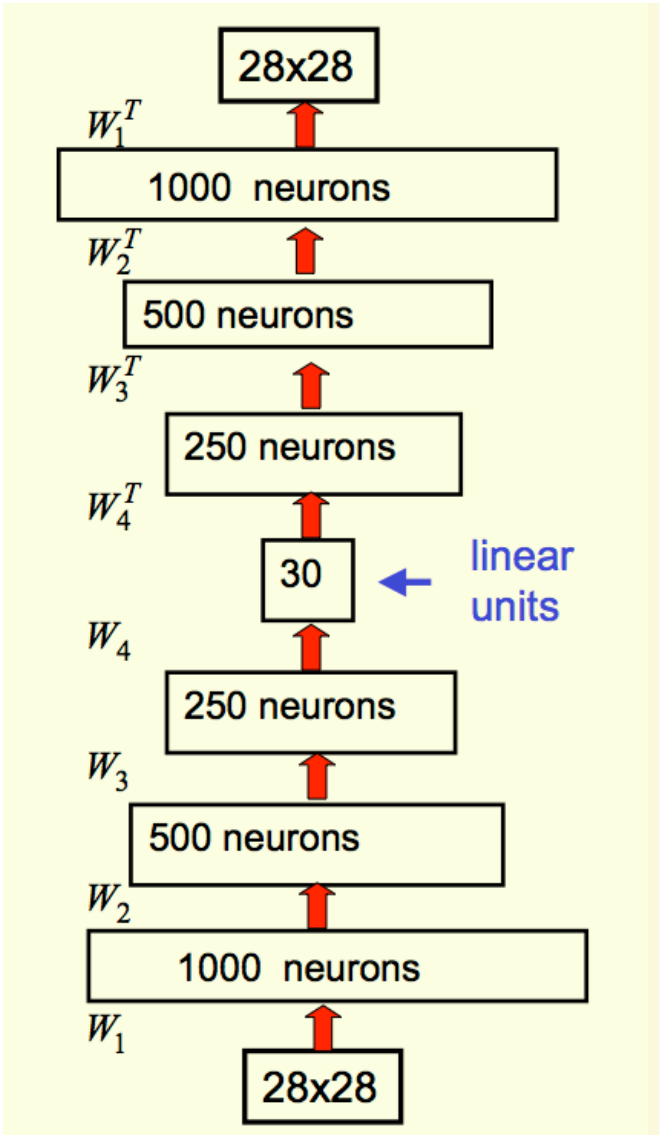
where $\mathcal{A}(X)$ is a linear mapping of the components of X (e.g. element-wise observations).

Can have ψ as the nuclear norm $=$ sum of singular values. This regularizer tends to promote low rank (in the same way as $\|x\|_1$ tends to promote sparsity of a vector x).

Alternatively: X is the sum of sparse matrix and a low-rank matrix. The element-wise 1-norm $\|X\|_1$ is useful in inducing sparsity.

Useful in recommender systems, e.g. Netflix, Amazon.

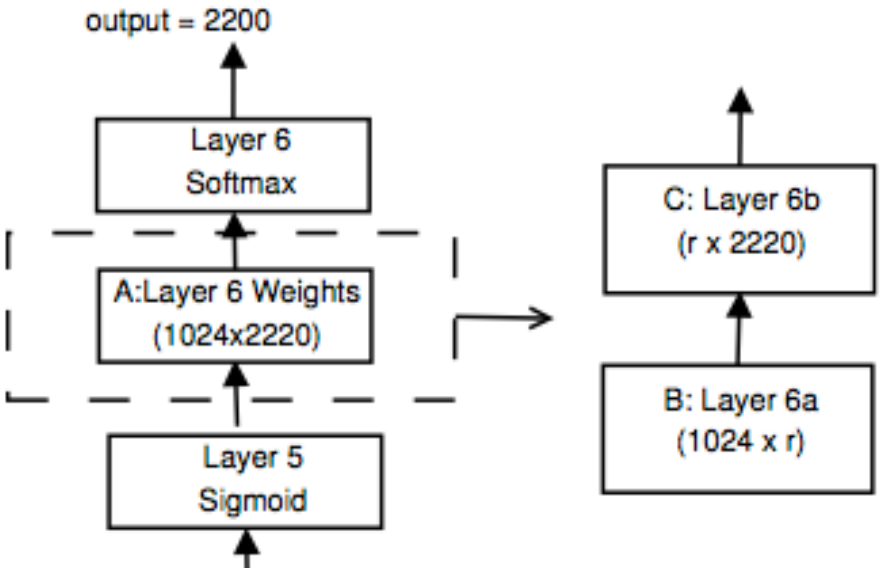
Deep Belief Networks



Deep Belief Nets / Neural Nets transform feature vectors prior to classification.

Example of a deep belief network for autoencoding (Hinton, 2007). Output (at top) depends on input (at bottom) of an image with 28×28 pixels. The unknowns are parameters of the matrices W_1, W_2, W_3, W_4; output is nonlinear in these parameters.

Deep Belief Networks



Objectives in learning problems based on these structures are

- separable: objective is composed of terms that each depend on one piece of data (e.g. one utterance, one character, one image)
- nonlinear, nonconvex: each layer is simple (linear transformation, sigmoid, softmax), but their composition is not.
- regularized: contains terms that impose structure. e.g. phoneme class depends on sounds that came before and after.

Nonconvexity is typically ignored at present, in algorithm design.

II. Stochastic Gradient

Algorithms that minimize a convex function f by either:

- Obtaining a cheap (unbiased) estimate of the gradient $\nabla f(x)$ at each step. Stochastic gradient.
- Evaluating just a few components of $\nabla f(x)$, selected randomly. Stochastic (block) coordinate descent.

This setup is extremely useful in data analysis:

- When f is a loss function defined over an entire data set, ∇f requires a scan of the full set...
- But we can get an unbiased estimate of ∇f by taking a small random sample of the data (one item, or a mini-batch).

I discuss mostly asynchronous parallel methods and their multicore implementations.

Stochastic Gradient

Typical form of f in learning problems:

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x),$$

where each f_i is convex, and depends on a single item of data.

In classical SG, choose index $i_k \in \{1, 2, \ldots, m\}$ uniformly at random at iteration k, set

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k), \quad \text{for some } \alpha_k > 0.$$

When f is strongly convex, the analysis of convergence of $E(\|x_k - x^*\|^2)$ is elementary (Nemirovski et al, 2009).

Averaging of iterates x_k and gradient estimates $\nabla f_{i_k}(x_k)$ (primal and dual averaging) can be applied to make behavior "smoother" and more robust.

Convergence of Classical SG

Suppose there are μ (convexity modulus) and M such that

$$[\nabla f(x) - \nabla f(x')]^T (x - x') \geq \mu \|x - x'\|^2,$$

$$\frac{1}{m} \sum_{i=1}^{m} \|\nabla f_i(x)\|^2 \leq M^2$$

for all x, x' of interest. Convergence obtained for the expected square error:

$$a_k := \frac{1}{2} E(\|x_k - x^*\|^2).$$

Elementary argument shows a recurrence:

$$a_{k+1} \leq (1 - 2\mu\alpha_k) a_k + \frac{1}{2} \alpha_k^2 M^2.$$

When we set $\alpha_k = 1/(k\mu)$, an inductive argument reveals a $1/k$ rate:

$$a_k \leq \frac{Q}{2k}, \qquad \text{for } Q := \max\left(\|x_1 - x^*\|^2, \frac{M^2}{\mu^2} \right).$$

Constant Step Size: $\alpha_k \equiv \alpha$

We can also obtain a "$1/k$ rate" for f strongly convex, for constant stepsize $\alpha_k \equiv \alpha$, when desired threshold ϵ for a_k is specified a priori.

From earlier analysis, have

$$a_{k+1} \leq (1 - 2\mu\alpha)a_k + \frac{1}{2}\alpha^2 M^2$$

which easily yields

$$a_k \leq (1 - 2\mu\alpha)^k a_0 + \frac{\alpha M^2}{4\mu}.$$

Given $\epsilon > 0$, choose α and K so that $a_k \leq \epsilon$ for all $k \geq K$. Choose so that both terms in the bound above are less than $\epsilon/2$:

$$\alpha := \frac{2\epsilon\mu}{M^2}, \qquad K := \frac{M^2}{4\epsilon\mu^2} \log\left(\frac{a_0}{2\epsilon}\right).$$

Parallel Stochastic Approximation

Several approaches tried for parallel stochastic approximation.

- **Dual Averaging (AIG):** Average gradient estimates evaluated in parallel on different cores. Requires message passing / synchronization (Dekel et al, 2011; Duchi et al, 2010).
- **Round-Robin (RR):** Cores evaluate ∇f_i in parallel and update centrally stored x in round-robin fashion. Requires synchronization (Langford et al, 2009).
- **Asynchronous:** HOGWILD!: Each core grabs the centrally-stored x and evaluates $\nabla f_i(x)$ for some random i, then writes the updates back into x (Niu et al, 2011). **Downpour SGD:** Similar idea for cluster (Dean et al, 2012).

HOGWILD!: Each processor runs independently:

1. Sample i_k from $\{1, 2, \ldots, m\}$;
2. Read current state of x from central memory, evalute $g := \nabla f_{i_k}(x)$;
3. **for** nonzero components g_v **do** $x_v \leftarrow x_v - \alpha g_v$;

HOGWILD! Convergence

- Updates can be "old" by the time they are applied, but we assume a bound of τ cycles on their age.
- Processors can overwrite each other's work, but sparsity of the ∇f_i helps — updates don't interfere too much with each other.

Niu, Recht, Ré, Wright (2011), simplified by Richtarik (2012).

Define quantities that capture the interconnectedness of the functions f_i:

- ρ_i = number of indices j such that f_i and f_j depend on overlapping components of x.
- $\rho = \sum_{i=1}^{m} \rho_i / m^2$: average rate of overlapping subvectors.

HOGWILD! Convergence (Richtarik)

Given $\epsilon \in (0, a_0 L)$, and setting

$$\alpha_k \equiv \frac{\mu\epsilon}{(1+2\tau\rho)LM^2m^2}$$

we have for

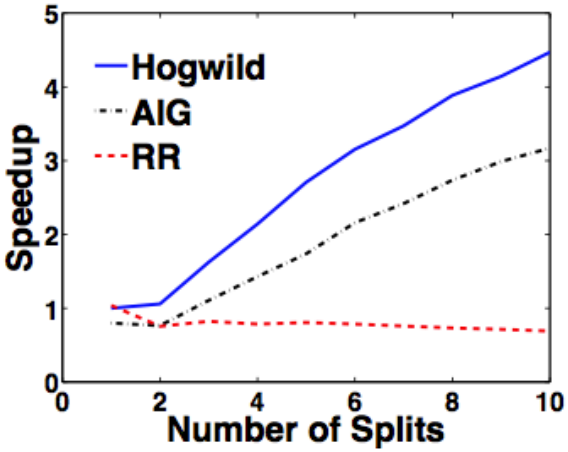
$$k \geq \frac{(1+2\tau\rho)LM^2m^2}{\mu^2\epsilon}\log\left(\frac{2La_0}{\epsilon}-1\right)$$

that

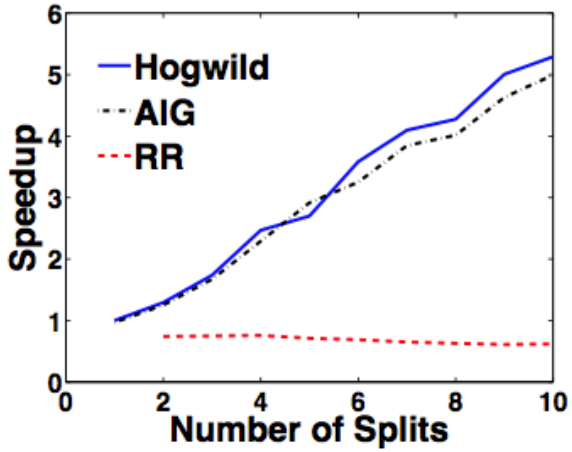
$$\min_{0\leq j\leq k} E(f(x_j)-f(x^*)) \leq \epsilon,$$

Recovers the sublinear $1/k$ convergence rate seen in regular SGD, with the delay τ and overlap measure ρ both appearing linearly.

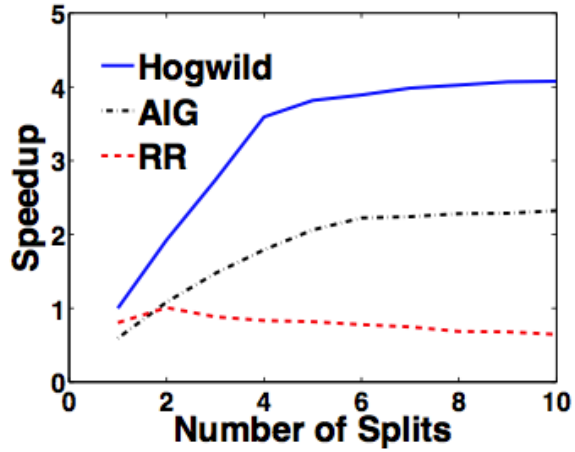
HOGWILD! Performance



SVM
RCV1



MC
Netflix



CUTS
Abdomen

HOGWILD! compared with averaged gradient (AIG) and round-robin (RR). Experiments run on a 12-core machine. (10 cores used for gradient evaluations, 2 cores for data shuffling.)

HOGWILD! Performance

	data set	size (GB)	ρ	Δ	time (s)	speedup
SVM	RCV1	0.9	4.4E-01	1.0E+00	10	4.5
	Netflix	1.5	2.5E-03	2.3E-03	301	5.3
MC	KDD	3.9	3.0E-03	1.8E-03	878	5.2
	JUMBO	30	2.6E-07	1.4E-07	9,454	6.8
	DBLife	0.003	8.6E-03	4.3E-03	230	8.8
CUTS	Abdomen	18	9.2E-04	9.2E-04	1,181	4.1

III. Kaczmarz for $Ax = b$.

Consider linear equations $Ax = b$, where the equations are consistent and matrix A is $m \times n$, not necessarily square or full rank. Write

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad \text{where } \|a_i\|_2 = 1,\ i = 1, 2, \ldots, m.$$

Iteration k of Kacmarz:

- Select row index $i_k \in \{1, 2, \ldots, m\}$ randomly with equal probability.
- Set

$$x_{k+1} \leftarrow x_k - (a_{i_k}^T x_k - b_{i_k}) a_{i_k}.$$

Equivalent to applying SG to

$$f(x) := \frac{1}{2m} \sum_{i=1}^{m} (a_i^T x - b_i)^2 = \frac{1}{2m} \|Ax - b\|_2^2,$$

with steplength $\alpha_k \equiv 1$.

Randomized Kaczmarz Convergence

Strohmer and Vershynin (2009) get a *linear* rate:

$$E\left[\|x_{k+1} - P(x_{k+1})\|_2^2 \,|\, x_k\right] \leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{\lambda_{\max}}\right) \|x_k - P(x_k)\|_2^2,$$

where P denotes projection onto solution subspace, and $\lambda_{\max}$ and $\lambda_{\min,\mathrm{nz}}$ denote largest and smallest nonzero eigenvvalues of $A^T A$.

We can obtain essentially the same linear rate by analyzing SG applied to the sum-of-squares form, following standard analysis but using special structure, namely, all f_i have zero gradients:

$$\nabla f_i(x^*) = a_i(a_i^T x^* - b_i) = 0, \quad i = 1, 2, \ldots, m.$$

Elementary SG analysis for $\min \|Ax - b\|^2$.

Assume A scaled so that $\|a_i\| = 1$ for all i. $\lambda_{\min,\mathrm{nz}}$ denotes minimum nonzero eigenvalue of $A^T A$. $P(\cdot)$ is projection onto solution set.

Choose steplength $\alpha_k \equiv 1$, we have

$$\begin{aligned}\frac{1}{2}\|x_{k+1} - P(x_{k+1})\|^2 &\leq \frac{1}{2}\|x_k - a_{i_k}(a_{i_k}^T x_k - b_{i_k}) - P(x_k)\|^2 \\ &= \frac{1}{2}\|x_k - P(x_k)\|^2 - \frac{1}{2}(a_{i_k}^T x_k - b_{i_k})^2.\end{aligned}$$

Taking expectations:

$$\begin{aligned}E\left[\frac{1}{2}\|x_{k+1} - P(x_{k+1})\|^2 \,|\, x_k\right] &\leq \frac{1}{2}\|x_k - P(x_k)\|^2 - \frac{1}{2}E\left[(a_{i_k}^T x_k - b_{i_k})^2\right] \\ &= \frac{1}{2}\|x_k - P(x_k)\|^2 - \frac{1}{2m}\|Ax_k - b\|^2 \\ &\leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m}\right)\frac{1}{2}\|x_k - P(x_k)\|^2.\end{aligned}$$

Asynchronous Kaczmarz

(Liu, Wright, in progress)

At each iteration j:

- Select i_j from $\{1, 2, \ldots, m\}$ with equal probability;
- Select t_j from the support of a_{i_j} with equal probability;
- Update:

$$x_{j+1} = x_j - \gamma \|a_{i_j}\|_0 (a_{i_j}^T x_{k(j)} - b_{i_j}) E_{t_j} a_{i_j},$$

where

- $k(j)$ is some iterate prior to j but no more than τ cycles old: $j - k(j) \leq \tau$;
- γ is a constant steplength;
- E_t is the $n \times n$ matrix of all zeros, except for 1 in the (t, t) location.

As in HOGWILD!, different cores run this process concurrently, updating an x accessible to all.

ARK Analysis: Linear Convergence

Analysis uses a similar model to HOGWILD!. Parameters include:

- $\chi := \max_{i=1,2,\ldots,m} \|a_i\|_0$ (maximum nonzero row count);
- $\alpha := \max_{i,t} \|a_i\|_0 \|AE_t a_i\| \leq \chi \|A\|$.

Idea of analysis: Choose some $\rho > 1$ (typically $\rho = 1 + O(1/m)$) and choose γ small enough to ensure that

$$\rho^{-1}\mathbb{E}(\|Ax_j - b\|^2) \leq \mathbb{E}(\|Ax_{j+1} - b\|^2) \leq \rho\mathbb{E}(\|Ax_j - b\|^2).$$

Not too much change to the residual on any one iteration. Hence, don't pay too much of a price for using old information in the asynchronous algorithm.

But don't want γ to be too tiny, otherwise overall progress is too slow. Stike a balance.

Gory Details

Choose any $\rho > 1$ and define γ via the following:

$$\phi = \min\left\{1, \frac{(\rho-1)(m\chi + 2\lambda_{\max}\tau\rho^\tau)}{2\rho^{\tau+1}\lambda_{\max}}, \sqrt{\frac{(\rho-1)(m\chi + 2\lambda_{\max}\tau\rho^\tau)^2}{\rho^\tau(m\alpha^2 + \lambda_{\max}^2\tau\rho^\tau)}}\right\}$$

$$\gamma = \frac{m\phi}{m\chi + 2\lambda_{\max}\tau\rho^\tau}$$

Then have

$$\rho^{-1}\mathbb{E}(\|Ax_j - b\|^2) \leq \mathbb{E}(\|Ax_{j+1} - b\|^2) \leq \rho\mathbb{E}(\|Ax_j - b\|^2)$$

$$\mathbb{E}(\|x_{j+1} - P(x_{j+1})\|^2) \leq \left(1 - \frac{(1-(1-\phi)^2)\lambda_{\min,\mathrm{nz}}}{m\chi + 2\lambda_{\max}\tau\rho^\tau}\right)\mathbb{E}(\|x_j - P(x_j)\|^2),$$

A Particular Choice

A particular choice of ρ leads to simplified results, in a reasonable regime.

Assume $2e\lambda_{\max}(\tau+1) \leq m$ and set

$$\rho = 1 + \frac{2e}{m}\lambda_{\max}.$$

Can show that $\phi = 1$ for this case, so expected convergence is

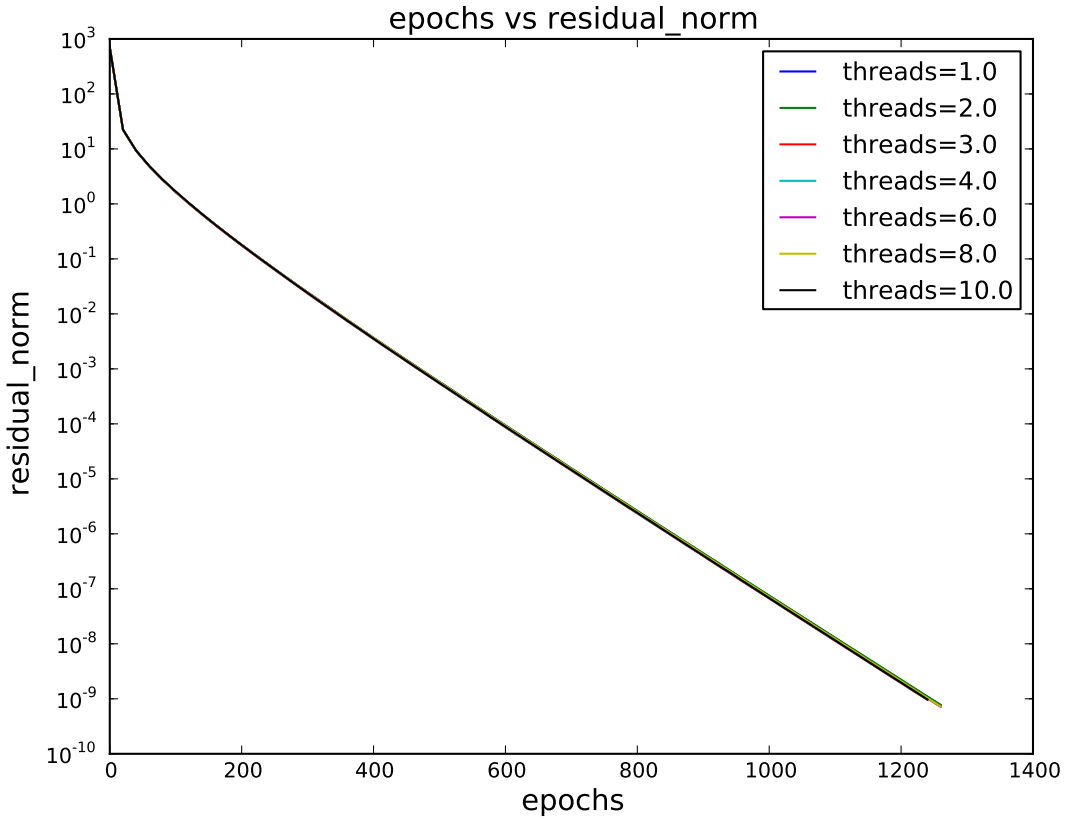
$$\begin{aligned}\mathbb{E}(\|x_{j+1} - P(x_{j+1})\|^2) &\leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m\chi + 2\lambda_{\max}\tau\rho^\tau}\right)\mathbb{E}(\|x_j - P(x_j)\|^2) \\ &\leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m\chi + 2e\lambda_{\max}\tau}\right)\mathbb{E}(\|x_j - P(x_j)\|^2).\end{aligned}$$

In the regime $2e\lambda_{\max}(\tau+1) \leq m$ considered here the delay τ doesn't really interfere with convergence rate.

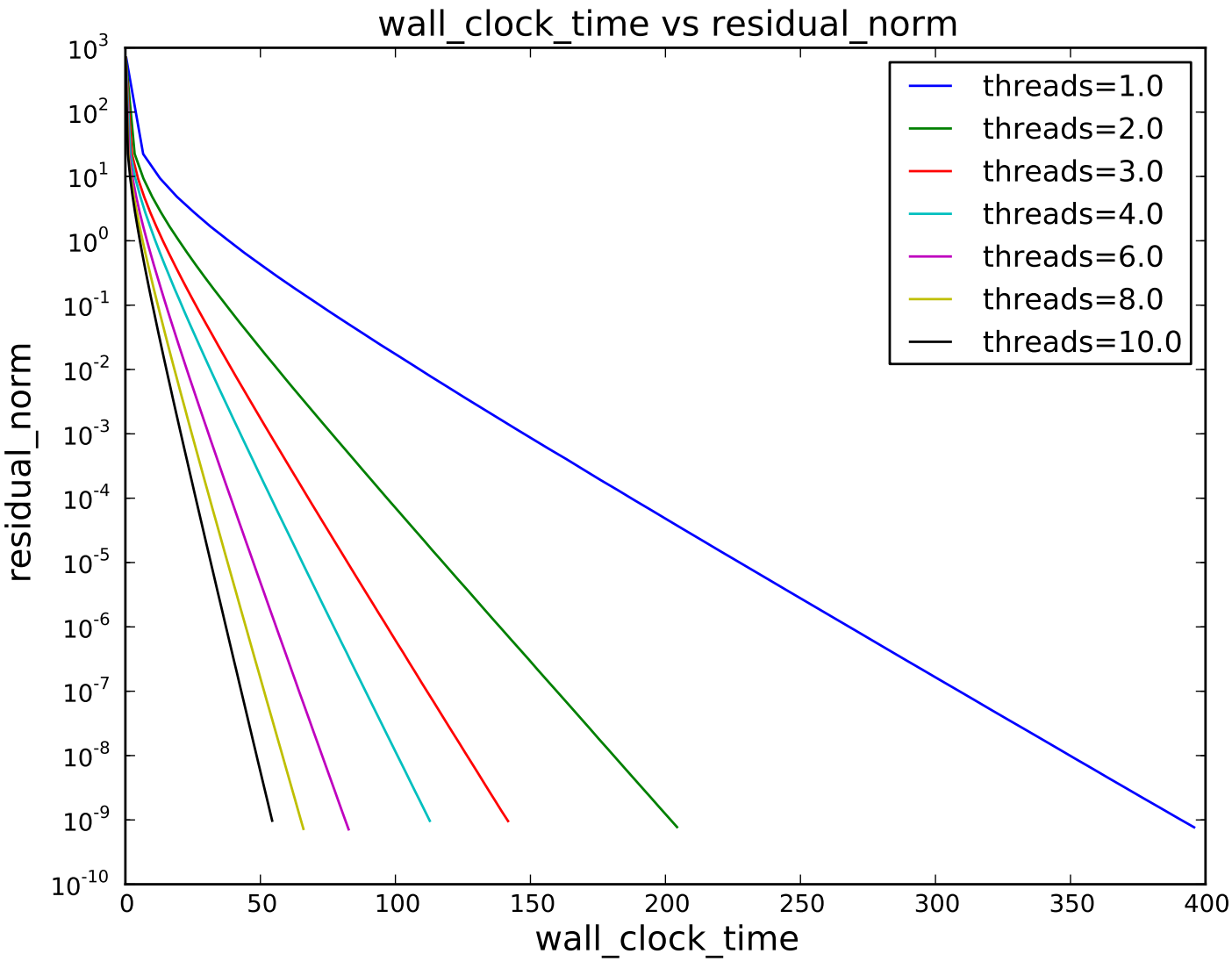
For random matrices A with unit rows have $\lambda_{\max} \sim (1 + O(m/n))$ to high probability.

Asynchronous Kaczmarz Results

Applied asynchronous parallel Kaczmarz to $Ax = b$ where A is $100,000 \times 80,000$ and 3% dense. (1.8GB total)



Asynchronous Kaczmarz: Timings



IV. Stochastic Coordinate Descent (SCD)

Richtarik and Takac (2012), Nesterov (2012)

Consider $\min f(x)$, where $f : \mathbb{R}^n \to \mathbb{R}^n$ is convex.

Block SCD: Partition components $\{1, 2, \ldots, n\}$ into m disjoint blocks denoted by $[i]$, $i = 1, 2, \ldots, m$. Denote:

- $E_{[i]}$ columns corresponding to $[i]$ from identity matrix I;
- L_i is partial Lipschitz constant on partition $[i]$:

$$\|\nabla_{[i]} f(x + E_{[i]} t) - \nabla_{[i]} f(x)\| \leq L_i \|t\|.$$

Iteration k:

- Choose partition $i_k \in \{1, 2, \ldots, m\}$ with equal probability;
- Set $x_{k+1} = x_k - E_{[i_k]} \nabla_{[i_k]} f(x_k) / L_{i_k}$.

SCD Convergence (Serial)

(Richtarik and Takac, 2012) For convex f, have high probability convergence of f to within a specified threshold ϵ of $f(x^*)$ in $O(1/\epsilon)$ iterations.
Given desired precision ϵ and error prob ρ, define

$$K := \frac{2n\mathcal{R}_L^2(x_0)}{\epsilon} \log \frac{f(x_0) - f^*}{\epsilon\rho},$$

(where $\mathcal{R}_L^2(x_0)$ is the radius of the level set in weighted norm $\|\cdot\|_L$). For $k \geq K$, have

$$P(f(x_k) - f^* \leq \epsilon) \geq 1 - \rho.$$

If f is strongly convex with respect to $\|\cdot\|_L$, with modulus μ_L in this norm, there is expected convergence at a linear rate:

$$E[f(x_k) - f^*] \leq \left(1 - \frac{\mu_L}{4n}\right)^k (f(x_0) - f^*).$$

Asynchronous SCD

Similar computation model to HOGWILD!: asynchronous with maximum delay τ. Consider single-coordinate form.

At each iteration j:

- Choose i_j with equal probability from $\{1, 2, \ldots, n\}$;
- Update the i_k component:

$$x_{j+1} = x_j - \gamma E_{i_j} \nabla f(x_{k(j)}),$$

where $k(j)$ is some iterate prior to j but no more than τ cycles old: $j - k(j) \leq \tau$. Here γ is a constant steplength.

Each core runs this process concurrently and asynchronously.

Asynchronous SCD Analysis

Richtarik (2012)

In the case of (single-coordinate) asynchronous SGD, given tolerance ϵ, set

$$\gamma \equiv \frac{2\theta\mu\epsilon}{(n+2\tau)L^2M^2}.$$

Then for

$$K \geq \frac{(n+2\tau)LM^2}{\mu^2\epsilon}\log\left(\frac{2LD_0}{\epsilon}-1\right)$$

we have a "morally $1/k$" rate:

$$\min_{0\leq j\leq K} E[f(x_j)-f^*] \leq \epsilon.$$

Here, μ (convexity modulus of f), M (uniform bound on $\|\nabla f(x)\|$), L (Lipschitz for ∇f) are defined as earlier.

Asynchronous SCD: Linear Convergence

(Liu, Wright, in progess)

Additional notation:

- L_i = coordinate Lipschitz constant: $|[\nabla f(x)]_i - [\nabla f(x + te_i)]_i| \leq L_i t$;
- $L_{\max} = \max_{i=1,2,\ldots,n} L_i$;
- $R = \sup_k \operatorname{dist}(x_k, \mathcal{S})$: maximum distance of iterates from solution set.

Motivation similar to ARK analysis: Choose some $\rho > 1$ and pick γ small enough to ensure that

$$\rho^{-1} \leq \frac{\mathbb{E}(\|\nabla f(x_{j+1})\|^2)}{\mathbb{E}(\|\nabla f(x_j)\|^2)} \leq \rho.$$

Not too much change in gradient over each iteration, so not too much price to pay for using old information, in the asynchronous setting.

But can still choose γ large enough to get a linear rate.

Gory Details (Strongly Convex f)

For any $\rho > 1$, define

$$\phi = \min \left\{ 1, \frac{(\rho - 1)(\sqrt{n} L_{\max} + 2L\tau\rho^{\tau})}{L \max\{2\rho^{\tau+1}, 2 + \frac{L\rho^{\tau}}{\sqrt{n} L_{\max}}\}} \right\}$$

and

$$\gamma = \frac{n\phi}{nL_{\max} + 2\sqrt{n}\tau\rho^{\tau} L}.$$

We have for any $j \geq 0$

$$\mathbb{E}(f(x_{j+1}) - f^*) \leq \left(1 - \frac{(1 - (1-\phi)^2)\mu}{nL_{\max} + 2\sqrt{n}\tau\rho^{\tau} L}\right)^{j+1} (f(x_0) - f^*).$$

A Particular Choice

Consider the regime in which

$$\tau + 1 \leq \frac{\sqrt{n} L_{\max}}{2eL},$$

and define

$$\rho = 1 + \frac{2eL}{\sqrt{n} L_{\max}}.$$

Thus $\rho^{\tau+1} \leq e$ and $\phi = 1$, and the rate simplifies to:

$$\mathbb{E}(f(x_{j+1}) - f^*) \leq \left(1 - \frac{\mu}{nL_{\max} + 2eL\sqrt{n}\tau}\right)^{j+1} (f(x_0) - f^*),$$

yielding expected convergence to precision ϵ in K iterations, where

$$K \geq \frac{|\log \epsilon|}{\mu}(nL_{\max} + 2eL\tau\sqrt{n}).$$

The regime on τ is somewhat restrictive, but the degradation as τ exceeds this bound (and ϕ drops below 1) is not too severe.

Gory Details (Weakly Convex f: $\mu = 0$)

Defining ϕ as above, have

$$\mathbb{E}(f(x_{j+1}) - f^*) \leq \frac{1}{(f(x_0) - f^*)^{-1} + \frac{1-(1-\phi)^2}{R^2(2nL_{\max}+4eL\sqrt{n}\tau)}(j+1)}.$$

Thus obtain expected convergence to precision ϵ in K iterations, where

$$K \geq \frac{R^2(2nL_{\max} + 4eL\sqrt{n}\tau)}{1-(1-\phi)^2}\left(\frac{1}{\epsilon} - \frac{1}{f(x_0) - f^*}\right).$$

Roughly "$1/k$" behavior.

SCD: Comments

Approaches above are extendible for

- separable constraints: $x \in \Omega$, $\Omega = \Omega_1 \times \Omega_2 \times \cdots$;
- separable regularizer: $f(x) + \lambda\psi(x)$, where $\psi(x) = \psi_1(x_{[1]}) + \psi_2(x_{[2]}) + \ldots$.
- blocks.

(Already done by Richtarik and Takac, in various settings.)

Another recent related paper: Avron, Druinsky, Gupta (last week): $Ax = b$, where A is symmetric positive definite. As well as the "HOGWILD!" model of asynchronous computation, they propose an "inconsistent read" model. Linear convergence.

Computations: SCD for QP with Bounds

Apply SCD to bound-constrained convex QP:

$$\min \frac{1}{2} x^T Q x + p^T x \text{ s.t. } l \leq x \leq u,$$

where Q is $n \times n$ symmetric positive definite.

These arise in the dual formulation of support vector machines (SVM) if the intercept term is omitted.

Interesting Q have a variety of structures:

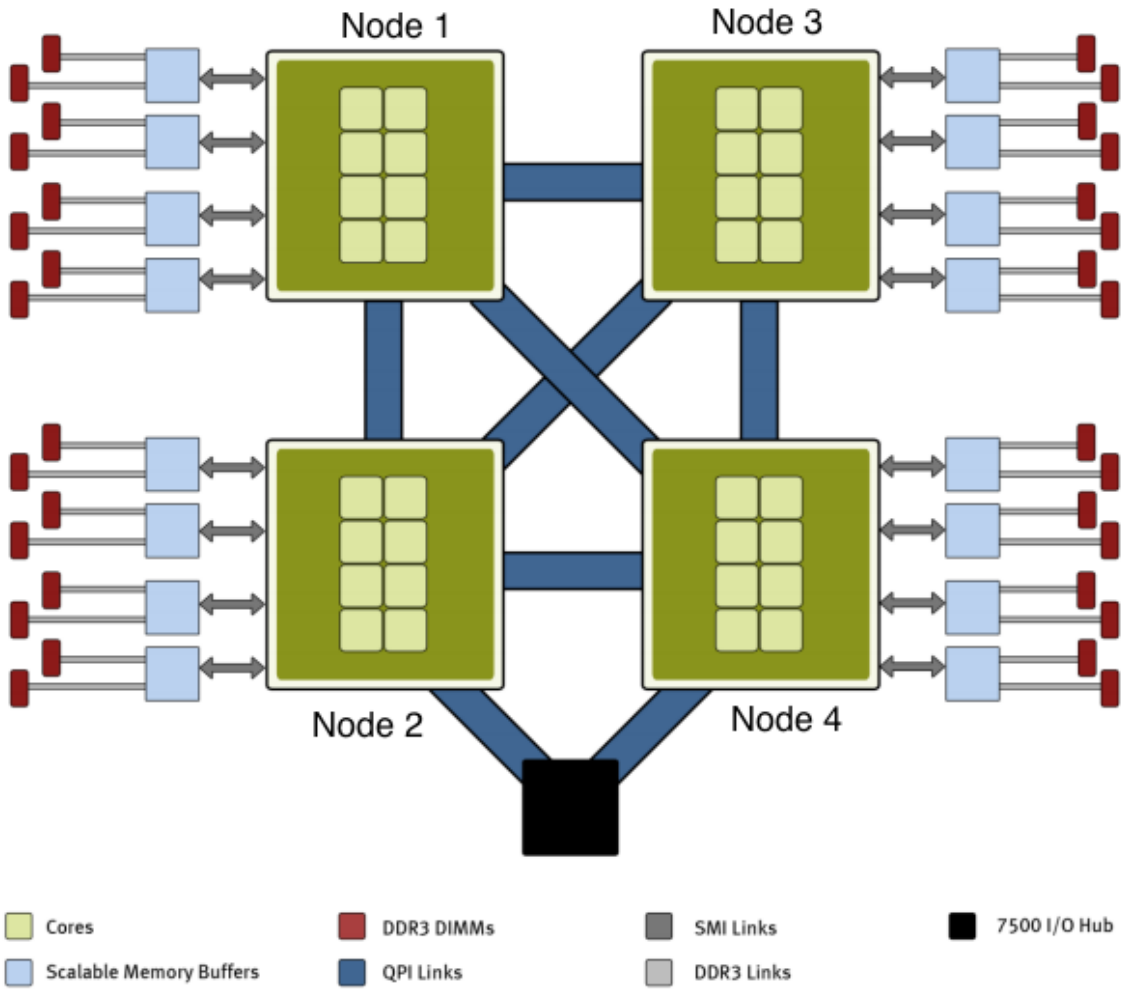
- fully dense (typical in SVM; may approximate as sparse + low-rank);
- sparse: general or structured;
- $Q = AA^T$ for some A of size $n \times r$. Can have $r \ll n$ and A dense, or A is itself sparse.

Asynchronous SCD for QP

- Use single components (not blocks);
- Project onto to the feasible interval $[l_i, u_i]$;
- don't sample with replacement — rather divide into "epochs." Each x_i, $i = 1, 2, \ldots, n$ updated exactly once in each epoch. Ordering is reshuffled between epochs.
- Multicore processor, with indices $i = 1, 2, \ldots, n$ distributed between cores.

while not converged **do**
  for $i \in \{1, 2, \ldots, n\}$ in parallel **do**
    Compute $g_i = Q_{i\cdot}x + p_i$;
    $x_i \leftarrow \text{mid}(l_i, x_i - g_i/Q_{ii}, u_i)$;
  end for
end while

Implemented on 4-socket, 40-core Intel Xeon



$Q = AA^T$: DIMM-WITTED: Lazy and Eager

SVM with linear kernel on RCV1 test set. Bound-constrained QP with $Q = AA^T$. Each row of A is the feature vector for a single item of data.

n = number of data vectors from RCV1.

- Eager: Q is precomputed and partitioned between sockets.
- Lazy: Use A, stored on every socket. Don't compute Q explicitly.

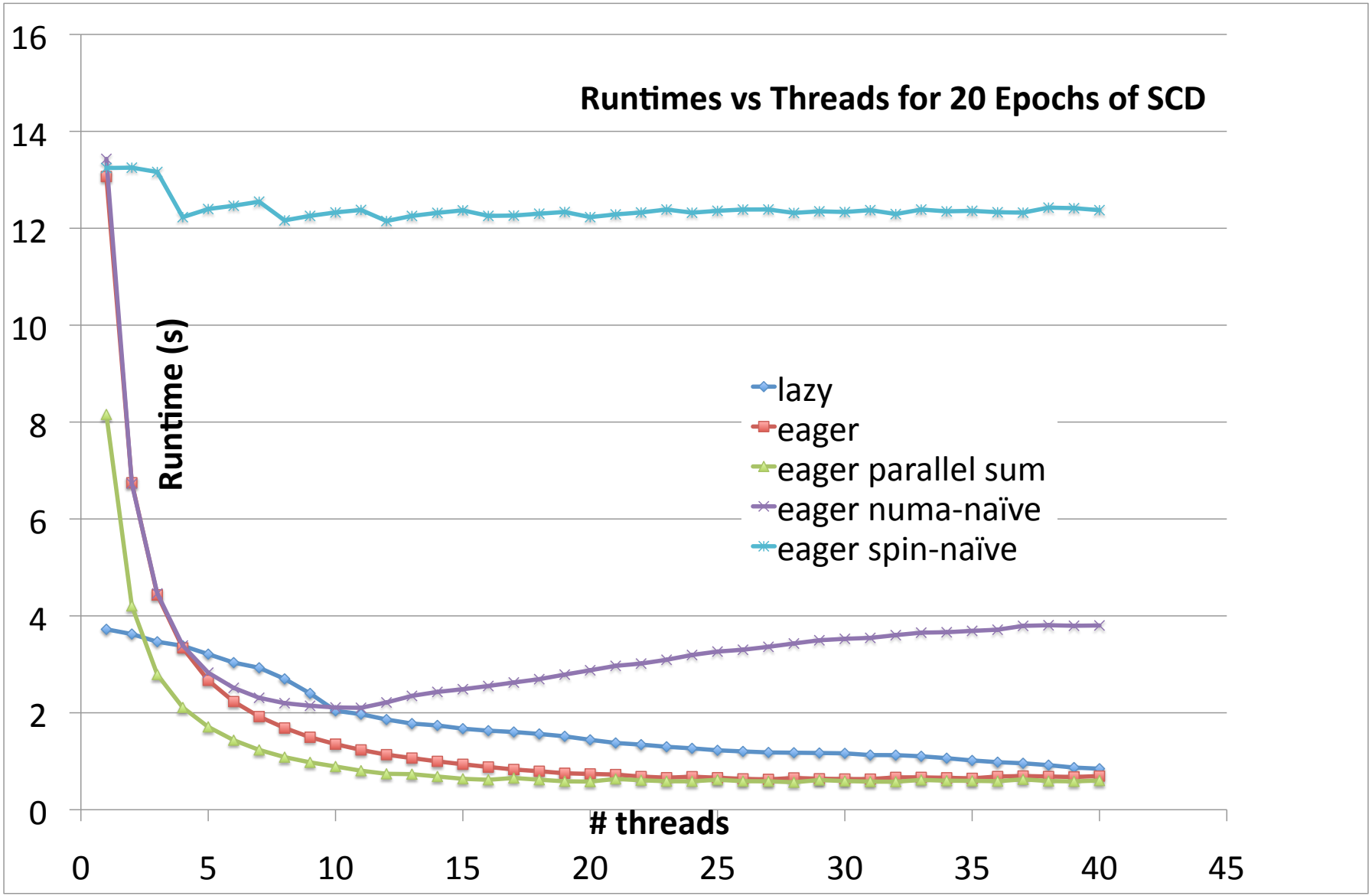
Eager details:

- Indices reshuffled between epochs, *within sockets* only.
- A core must read latest x to take a step, but most components it needs are already in cache. Needs to fetch only those components recently changed.

SCD Variants Plotted

Do 20 epochs of SCD on the problem with 1.2 GB of data ($n = 12596$).

- **Lazy**: Store A (on every socket), not Q.
- **Eager**: Q precomputed (6 seconds approx).
- **Eager: Spin-Naive**: Lock x while reading and writing.
- **Eager: NUMA-Naive**: Cores select index i to update without regard to where $Q_{i:}$ is stored — possibly need to fetch it from another socket.
- **Parallel Sum**: Speed limit: simply sum the elements of Q, 20 times. "Eager" SCD cannot be faster than this.


Runtimes vs Threads for 20 Epochs of SCD
Runtime (s)
threads
lazy
eager
eager parallel sum
eager numa-naïve
eager spin-naïve
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25
30
35
40
45

RCV1 Results

Table: Time to convergence for RCV1 probems, on 32 Cores.

Sl.	Vars	Data	Solve time (secs)				
			Cplex(B)	Cplex(S)	Gurobi(B)	Gurobi(S)	SCD
1	123	0.2MB	0.031	0.065	0.081	0.03	0.05 ($\epsilon = 10^{-5}$)
2	12596	1.2GB	5882	1691	3002	2708	6.9 ($\epsilon = 10^{-5}$)

- S = Simplex, B = Barrier
- Time limit: 2 hours

Conclusions

Have considered mostly asynchronous, stochastic algorithms for partially separable optimization, and linear equations.

Convergence theory supports computational observations that good speedup is possible.

Asynchronicity is key to speedup on modern architectures (see Bill Gropp, SIAM CS&E Plenary, 2013).

These techniques may not be complete solutions in themselves, but may form part of a solution strategy for particular applications.