SMART: The Stochastic Monotone Aggregated Root-Finding Algorithm

Damek Davis[1]

Department of Mathematics
University of California, Los Angeles/
School of Operations Research and Information Engineering
Cornell University

[1] http://www.math.ucla.edu/~damek

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \; f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(a_i^T x)$$

- The **empirical risk minimization problem** (ERM)
- $A = (a_1, \ldots, a_n) \in \mathbb{R}^{m \times n}$.
- $n =$ number of training examples
- $m =$ number of features.

- **Nice Properties:**
  - $f_i : \mathbb{R} \to \mathbb{R}$ smooth, one dimensional, convex
  - $\nabla(f_i \circ a_i^T) : \mathbb{R}^m \to \mathbb{R}^m$ in one dimensional space

$$\nabla(f_i \circ a_i^T)(x) = a_i f_i'(a_i^T x) \in \text{Range}(a_i).$$

  - So to compute gradient, need one inner product, one scalar derivative.

- **Gradient Descent:** (fast; high per iteration cost; low memory)

$$x^{k+1} = x^k - \frac{\gamma}{n} \sum_{i=1}^{n} a_i f_i'(a_i^T x^k)$$

  - Need to compute $A^T x^k$ and all scalar gradients, then sum them together.

- **Gradient Descent:** (fast; high per iteration cost; low memory)

$$x^{k+1} = x^k - \frac{\gamma}{n} \sum_{i=1}^{n} a_i f_i'(a_i^T x^k)$$

  - Need to compute $A^T x^k$ and all scalar gradients, then sum them together.

- **Stochastic Gradient** (slow; low per iteration cost; low memory)

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma_k a_{i_k} f_{i_k}'(a_{i_k}^T x)$$

  - Need $\gamma_k \to 0$, which can be slow!

- **Stochastic Variance Reduced Gradient (SVRG):** (fast; some high cost iterations, but mostly low; low memory)

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} f'_{i_k}(a_{i_k}^T \phi^k) + \frac{1}{n} \sum_{i=1}^{n} a_i f'_i(a_i^T \phi^k) \right)$$

$$\phi^{k+1} = \begin{cases} x^k & \text{if } k \equiv 0 \mod \tau; \\ \phi^k & \text{otherwise.} \end{cases}$$

- Every τ iterations, recompute:

$$\nabla f(x^k) = \frac{1}{n} \sum_{i=1}^{n} a_i f'_i(a_i^T x^k)$$

  otherwise, use the $\nabla f(\phi^k)$.
- Two derivatives computed per iteration.
- $\nabla f(x^k)$ stored $\implies$ memory is m-dimensional vector
- Strong convexity assumed.

- **Finito:** (fast; low per iteration cost; HIGH memory)

Sample $i_k \in \{1, \dots, n\}$ uniformly

$$x_i^{k+1} = \begin{cases} \frac{1}{n} \sum_{l=1}^{n} \left(x_l^k - \gamma a_l f_l'(a_l^T x_l^k) \right) & \text{if } i = i_k. \\ x_i^k & \text{otherwise.} \end{cases}$$

  - Need to store points $x_1^k, \dots, x_n^k$ AND gradients $f_l'(a_1^T x_1^k), \dots, f_n'(a_n^T x_n^k)$.
  - Strong convexity assumed.

- **Stochastic Average Gradient (SAG):** (fast; low per iteration cost; low memory)

Sample $i_k \in \{1, \dots, n\}$ uniformly

$$x^{k+1} = x^k - \frac{\gamma}{n}\left(a_{i_k} f'_{i_k}(a_{i_k}^T x^k) + \sum_{i \neq i_k} a_i z_i^k\right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

  - Memory is n-dimensional vector $(z_1^k, \dots, z_n^k)$.
  - Biased gradient:

$$\mathbb{E}\left[a_{i_k} f'_{i_k}(a_{i_k}^T x) + \frac{1}{n}\sum_{i \neq i_k} a_i z_i^k \mid x^k, \dots, x^0\right] = \frac{1}{n}\nabla f(x^k) + \left(1 - \frac{1}{n}\right)\sum_{i=1}^n a_i z_i^k.$$

  - COMPLICATED PROOF.
  - First incremental method where strong convexity NOT assumed.

- **SAGA:** (fast; low per iteration cost; low memory)

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} z_{i_k}^k + \frac{1}{n} \sum_{i=1}^n a_i z_i^k \right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

- Memory is n-dimensional vector $(z_1^k, \ldots, z_n^k)$.
- Unbiased gradient.
- Relatively simple proof
- Strong convexity NOT assumed.

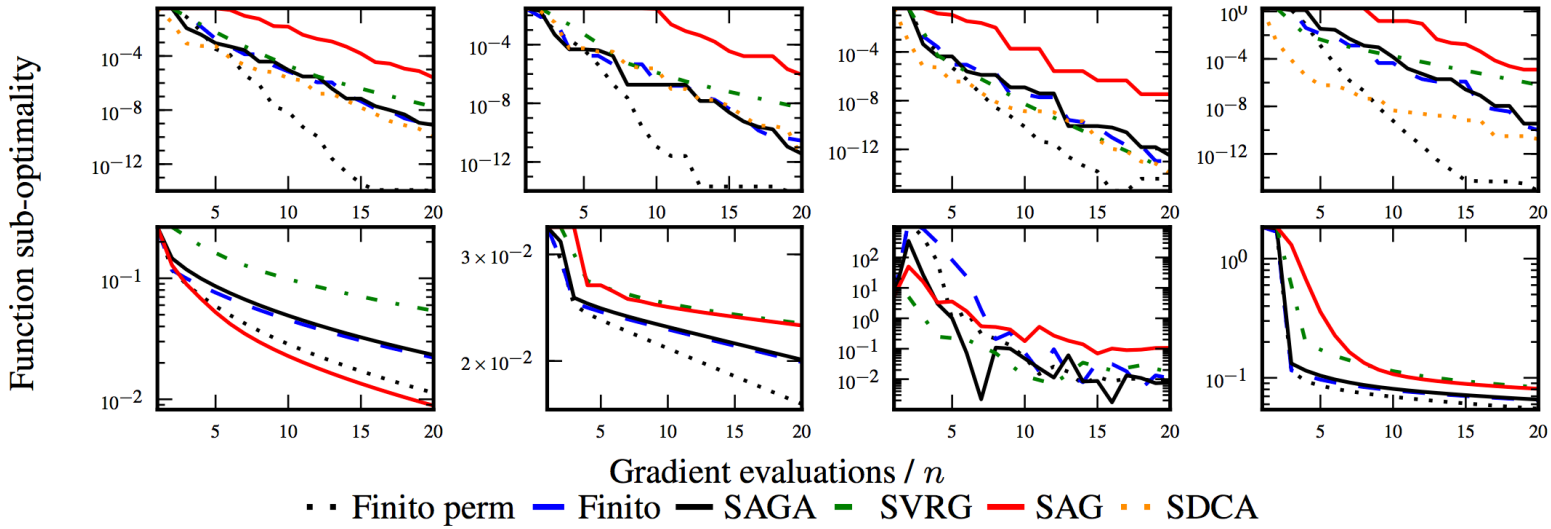

Figure 2: From left to right we have the MNIST, COVTYPE, IJCNN1 and MILLIONSONG datasets. Top row is the L2 regularised case, bottom row the L1 regularised case.

- Image stolen from SAGA paper
- SDCA only solves ℓ_2 regularized problem, so we ignored it.
- **Point:** All perform about the same, besides *Finito perm*, which isn't guaranteed to converge.

- **Today:**
- SMART extends incremental aggregated gradient and coordinate descent methods.
- SMART solves the ERM problem, and this seems to be its the most effective use, but it can go much further.
- In addition, SMART recovers: SAGA, Finito, SVRG, and SDCA.
- SAGA seems to be the catalyst for a lot the other methods, so **let's extend SAGA as much as possible.**

- **What if data matrix is sparse?**
- The gradient $a_{i_k} f'_{i_k}(a_{i_k}^T x^k)$ only has a few nonzero components.
- SAGA requires dense update for x^k because the sum is dense

$$\sum_{i=1}^{m} a_i z_i^k$$

- We should only update components of x that are in the support of a_{i_k}, i.e., apply mask to the gradient sum.
- Makes gradient biased, no reason it should work.

- To avoid biased gradients, need to scale components of updates.
  - Let $\mathsf{C}_i \subseteq \{1, \ldots, m\}$ be the support of a_i.
  - Let $e_{\mathsf{C}_i} = \sum_{j \in C_i} e_i \leftarrow$ component mask.
  - Let $\mathbf{q}$ and $\mathbf{p}_i$ be vectors of probabilities (easy to precompute)

- To avoid biased gradients, need to scale components of updates.
  - Let $\mathsf{C}_i \subseteq \{1, \dots, m\}$ be the support of a_i.
  - Let $e_{\mathsf{C}_i} = \sum_{j \in C_i} e_i \leftarrow$ component mask.
  - Let $\mathbf{q}$ and $\mathbf{p}_i$ be vectors of probabilities (easy to precompute)

- **Sparse SAGA:**

Sample $i_k \in \{1, \dots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma e_{\mathsf{C}_{i_k}} \odot \mathbf{q} \odot \left(\mathbf{p}_i \odot (a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} z_{i_k}^k) + \frac{1}{n} \sum_{i=1}^n a_i z_i^k \right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

- The sparse update equation is a block coordinate update equation.
- **Block Coordinate SAGA:**

Sample $i_k \in \{1, \ldots, n\}$ uniformly and $\mathsf{S}_k \subseteq \{1, \ldots, m\}$ arbitrarily

$$x^{k+1} = x^k - \gamma e_{\mathsf{S}_k} \odot \mathbf{q} \odot \left(\mathbf{p}_i \odot (a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} z_{i_k}^k) + \frac{1}{n} \sum_{i=1}^{n} a_i z_i^k \right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

- Coordinates and the gradient can be coupled.
- Only one function $\implies$ recover block-coordinate descent.

- We only compute one gradient per iteration, but we can gain a bit in performance if we compute a few more.
- Introducing the *trigger graph*: $G = (V, E)$.
  1. $V = \{1, \ldots, n\}$
  2. $E \subseteq V \times V$.
- We say that index i in V *triggers* i in V provided (i, i') is in E.

- We only compute one gradient per iteration, but we can gain a bit in performance if we compute a few more.
- Introducing the *trigger graph*: $G = (V, E)$.
  1. $V = \{1, \ldots, n\}$
  2. $E \subseteq V \times V$.
- We say that index i in V *triggers* i in V provided (i, i') is in E.
- **Minibatching SAGA:**

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} z_{i_k}^k + \frac{1}{n} \sum_{i=1}^{n} a_i z_i^k \right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i_k \textit{ triggers } i; \\ z_i^k & \text{otherwise.} \end{cases}$$

- Improves theoretical convergence rate and practical performance.

- What if we're not solving ERM problem, but we solve

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

- Then SAGA becomes a high memory method!

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(\nabla f_{i_k}(x^k) - y_{i_k}^k + \frac{1}{n} \sum_{i=1}^{n} y_i^k \right)$$

$$y_i^{k+1} = \begin{cases} \nabla f_{i_k}(x^k) & \text{if } i = i_k; \\ y_i^k & \text{otherwise.} \end{cases}$$

- What if we're not solving ERM problem, but we solve

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

- Then SAGA becomes a high memory method!

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(\nabla f_{i_k}(x^k) - y_{i_k}^k + \frac{1}{n} \sum_{i=1}^{n} y_i^k \right)$$

$$y_i^{k+1} = \begin{cases} \nabla f_{i_k}(x^k) & \text{if } i = i_k; \\ y_i^k & \text{otherwise.} \end{cases}$$

- **Question:** Instead of saving individual gradients, can we just store the sum $\frac{1}{n} \sum_{i=1}^{n} y_i^k$, and periodically recompute it?

- Randomized delay ϵ_k + *complete trigger graph* = **SVRG clone**

Sample $i_k \in \{1, \dots, n\}$ uniformly and $\epsilon_k \in \{0, 1\}$

$$x^{k+1} = x^k - \gamma \left(\nabla f_{i_k}(x^k) - y_{i_k}^k + \frac{1}{n} \sum_{i=1}^{n} y_i^k \right)$$

$$y_i^{k+1} = y_i^k + \epsilon_k (\nabla f_{i_k}(x^k) - y_i^k).$$

- Randomized delay ϵ_k + *complete trigger graph* = **SVRG clone**

Sample $i_k \in \{1, \dots, n\}$ uniformly and $\epsilon_k \in \{0, 1\}$

$$x^{k+1} = x^k - \gamma \left(\nabla f_{i_k}(x^k) - y_{i_k}^k + \frac{1}{n} \sum_{i=1}^n y_i^k \right)$$

$$y_i^{k+1} = y_i^k + \epsilon_k (\nabla f_{i_k}(x^k) - y_i^k).$$

- **The trick:** $y_i^k = \nabla f_i(\phi^k)$ for old iterate x^k.

Sample $i_k \in \{1, \dots, n\}$ uniformly and $\epsilon_k \in \{0, 1\}$

$$x^{k+1} = x^k - \gamma \left(\nabla f_{i_k}(x^k) - \nabla f_{i_k}(\phi^k) + \frac{1}{n} \sum_{i=1}^n \nabla f_i(\phi^k) \right)$$

$$\phi^k = x^k + \epsilon_k (x^k - \phi^k).$$

- On average gradient, full gradient computed once every $\mathbb{E}[\epsilon_k]$ iterates (can be chosen however you want.)

- **Back to ERM problem....**
- Can also add importance sampling, which increases the range of step sizes we can take.

Without importance sampling: $\gamma \leq (2 \max\{L_i\})^{-1}$

Without importance sampling: $\gamma \leq \left(\frac{2}{n} \sum_i L_i \right)^{-1}$

- **Back to ERM problem....**
- Can also add importance sampling, which increases the range of step sizes we can take.

$$\text{Without importance sampling: } \gamma \leq (2\max\{L_i\})^{-1}$$

$$\text{Without importance sampling: } \gamma \leq \left(\frac{2}{n}\sum_i L_i\right)^{-1}$$

- **SAGA with Importance Sampling:**

Sample $i_k \in \{1, \ldots, n\}$ arbitrarily

$$x^{k+1} = x^k - \gamma\left(\mathbf{p}_i \odot (a_{i_k} f'_{i_k}(a_{i_k}^T x^k) - a_{i_k} z_{i_k}^k) + \frac{1}{n}\sum_{i=1}^n a_i z_i^k\right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^k) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

- Improves theoretical convergence rate and practical performance.

- **These algorithms are serial**; only one gradient is touched per iteration.
- Let's parallelize: choose $d_k \in \{1, \ldots, \tau\}^m$ and $e_k^i \in \{1, \ldots, \tau\}$. Set

$$\boxed{x^{k-d_k} = (x_1^{k-d_{k,1}}, \ldots, x_m^{k-d_{k,m}})}$$

- **These algorithms are serial**; only one gradient is touched per iteration.
- Let's parallelize: choose $d_k \in \{1, \ldots, \tau\}^m$ and $e_k^i \in \{1, \ldots, \tau\}$. Set

$$\boxed{x^{k-d_k} = (x_1^{k-d_{k,1}}, \ldots, x_m^{k-d_{k,m}})}$$

- **Asynchronous SAGA:**

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k - \gamma \left(a_{i_k} f'_{i_k}(a_{i_k}^T x^{k-d_k}) - a_{i_k} z_{i_k}^{k-e_k^{i_k}} + \frac{1}{n} \sum_{i=1}^n a_i z_i^{k-e_k^{i_k}} \right)$$

$$z_i^{k+1} = \begin{cases} f'_{i_k}(a_{i_k}^T x^{k-e_k^{i_k}}) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

- Can be combined with block coordinate updates.
- It's like running serial SAGA on n different processors, without ever syncing them up.

- These algorithms are nice, but they're lacking generality.
- Recall that SAGA and the other algorithms solve

$$\text{find } x \in \mathcal{H} \text{ such that: } \sum_{i=1}^{n} \nabla f_i(x) = 0$$

- SMART solves the following **Root-Finding Problem**:

$$\text{find } x \in \mathcal{H} \text{ such that: } S(x) = \frac{1}{n} \sum_{i=1}^{n} S_i(x) = 0$$

where $S_i : \mathcal{H} \to \mathcal{H}$ are gradient-like.

- **The Coherence Condition:** $(\exists \beta_{ij} > 0) : (\forall x \in \mathcal{H}), (\forall x^* \in \mathrm{zer}(S))$

$$\sum_{j=1}^{m} \sum_{i=1}^{n} \beta_{ij} \|(S_i(x))_j - (S_i(x^*))_j\|_j^2 \leq \langle S(x), x - x^* \rangle.$$

- **The Coherence Condition:** $(\exists \beta_{ij} > 0) : (\forall x \in \mathcal{H}), (\forall x^* \in \operatorname{zer}(S))$

$$\sum_{j=1}^{m} \sum_{i=1}^{n} \beta_{ij} \|(S_i(x))_j - (S_i(x^*))_j\|_j^2 \leq \langle S(x), x - x^* \rangle.$$

- Smooth convex functions satisfy

$$(\forall x, y \in \mathcal{H}) \qquad \frac{1}{L_i} \|\nabla f_i(x) - \nabla f_i(y)\|^2 \leq \langle \nabla f_i(x) - \nabla f_i(y), x - y \rangle.$$

  if ∇f_i is L_i-Lipschitz.

- $\implies$ property can be summed together for multiple smooth functions:

$$\sum_{i=1}^{n} \frac{1}{nL_i} \|\nabla f_i(x) - \nabla f_i(x^*)\|^2 \leq \langle \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x), x - x^* \rangle.$$

  if $\sum_{i=1}^{n} \nabla f_i(x^*) = 0$.

- **Proximal operators:** $(\forall x, y \in \mathcal{H})$

$$\begin{aligned}&\|(I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y)\|^2 \\ &\qquad \leq \langle (I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y), x - y \rangle.\end{aligned}$$

(for smooth and nonsmooth f)

- **Proximal operators:** $(\forall x, y \in \mathcal{H})$

$$\|(I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y)\|^2 \\ \leq \langle (I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y), x - y \rangle.$$

(for smooth and nonsmooth f)

- **Projection operators:** $(\forall x, y \in \mathcal{H})$

$$\|(I - P_C)(x) - (I - P_C)(y)\|^2 \leq \langle (I - P_C)(x) - (I - P_C)(y), x - y \rangle.$$

(for closed convex sets)

- **Proximal operators:** $(\forall x, y \in \mathcal{H})$

$$\begin{aligned}&\|(I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y)\|^2 \\ &\qquad \leq \langle (I - \mathbf{prox}_{\gamma f})(x) - (I - \mathbf{prox}_{\gamma f})(y), x - y \rangle.\end{aligned}$$

(for smooth and nonsmooth f)

- **Projection operators:** $(\forall x, y \in \mathcal{H})$

$$\|(I - P_C)(x) - (I - P_C)(y)\|^2 \leq \langle (I - P_C)(x) - (I - P_C)(y), x - y \rangle.$$

(for closed convex sets)

- **Subgradient projectors:** $(\forall x \in \mathcal{H}), (\forall x^* \in [f \leq 0])$

$$\left\| \frac{[f(x)]_+}{\|g(x)\|^2} g(x) \right\|^2 \leq \langle \frac{[f(x)]_+}{\|g(x)\|^2} g(x), x - x^* \rangle.$$

where $g(x) \in \partial f(x)$ is a *subgradient selector*.

Algorithm (SMART)

Let $\{\lambda_k\}_{k\in\mathbb{N}}$ be a sequence of stepsizes. Choose $x^0 \in \mathcal{H}$ and $y_1^0, \ldots, y_n^0 \in \mathcal{H}$ arbitrarily except that $y_{i,j}^0 = 0$ if $\mathbf{S}_{ij}^* = 0$. Then for $k \in \mathbb{N}$, perform the following three steps:

1. ***Sampling.*** *choose a set of coordinates S_k, an operator index i_k, and dual update decision ϵ_k.*
2. ***Primal update:*** *set*

$$(\forall j \in \mathsf{S}_k)\ x_j^{k+1} = x_j^k - \frac{\lambda_k}{q_j m n}\left(\frac{1}{p_{ij}}\left((S_{i_k}(x^{k-d_k}))_j - y_{i_k,j}^{k-e_k^{i_k}}\right) + \sum_{i=1}^{n} y_{i,j}^{k-e_k^i}\right)$$

$$(\forall j \notin \mathsf{S}_k)\ x_j^{k+1} = x_j^k.$$

3. ***Dual update:*** *If i_k triggers i, set*

$$\begin{aligned} \left(\forall j \in \mathsf{S}_k \text{ with } \mathbf{S}_{ij}^* \neq 0\right) \qquad & y_{i,j}^{k+1} = y_{i,j}^k + \epsilon_k\left((S_i(x^{k-d_k}))_j - y_{i,j}^k\right) \\ (\forall j \notin \mathsf{S}_k) \qquad & y_{i,j}^{k+1} = y_{i,j}^k. \end{aligned}$$

Otherwise, set $y_{i,j}^{k+1} = y_{i,j}^k$.

- Linear feasibility problem:

$$\text{Find } x \in \mathcal{H} \text{ such that } Ax = b$$

- **Randomized Asynchronous Kaczmarz algorithm:**

Sample $i_k \in \{1, \ldots, n\}$ uniformly

$$x^{k+1} = x^k + \lambda(b_{i_k} - \langle a_{i_k}, x^{k-d_k} \rangle) a_{i_k}$$

- Here we used $C_i = \{x \mid \langle a_i, x \rangle = b_i\}$ and

$$S_i := (I - P_{C_i}).$$

- No memory needed precisely because $S_i(x^*) = 0$ at any solution.

- Nonsmooth regularization of ERM?

$$\underset{x \in \mathcal{H}}{\text{minimize}} \; g(x) + \frac{1}{N} \sum_{i=1}^{N} f_i(a_i^T x),$$

- Nonsmooth regularization of ERM?

$$\underset{x\in\mathcal{H}}{\text{minimize}}\ g(x)+\frac{1}{N}\sum_{i=1}^{N} f_i(a_i^T x),$$

- Operators (that satisfy the coherence condition)

$$S_i=\frac{1}{L_i\|a_i\|^2 N}a_i\nabla f_i\circ a_i^T\mathbf{prox}_{L^{-1}g} \qquad i=1,\dots,N;$$

$$S_{N+1}=(I-\mathbf{prox}_{L^{-1}g}),$$

- Roots $x^*\in\mathrm{zer}(S)$ are not minimizers, but $\mathbf{prox}_{L^{-1}g}(x)$ is a minimizer.

- Nonsmooth regularization of ERM?

$$\underset{x\in\mathcal{H}}{\text{minimize}}\ g(x)+\frac{1}{N}\sum_{i=1}^{N} f_i(a_i^T x),$$

- Operators (that satisfy the coherence condition)

$$S_i=\frac{1}{L_i\|a_i\|^2 N}a_i\nabla f_i\circ a_i^T\mathbf{prox}_{L^{-1}g} \qquad i=1,\dots,N;$$

$$S_{N+1}=(I-\mathbf{prox}_{L^{-1}g}),$$

- Roots $x^*\in\operatorname{zer}(S)$ are not minimizers, but $\mathbf{prox}_{L^{-1}g}(x)$ is a minimizer.

- Every time we evaluate S_i, we have to evaluate $\mathbf{prox}_{L^{-1}g}$, make the trigger graph a star and we always update the $(N+1)$rst dual variable.

- **Asynchronous Proximal SAGA:**

Sample $i_k \in \{1, \dots, N+1\}$ with $P(i_k = i) = \begin{cases} \frac{1}{2N} & \text{if } i < N; \\ \frac{1}{2} & \text{if } i = N. \end{cases}$

if $i_k < N$

$$x^{k+1} = x^k - \frac{1}{\lambda(N+1)} \left(\frac{1}{p_{i_k}} (S_i(x^{k-d_k}) - a_i z_i^{k-e_k^{i_k}}) + y_{N+1}^{k-e_k^{N+1}} + \sum_{i=1}^{N} a_i z_i^{k-e_k^i} \right);$$

else

$$x^{k+1} = x^k - \frac{1}{\lambda(N+1)} \left(\frac{1}{p_{N+1}} (S_{N+1}(x^{k-d_k}) - y_{N+1}^{k-e_k^{N+1}}) + y_{N+1}^{k-e_k^{N+1}} + \sum_{i=1}^{N} a_i z_i^{k-e_k^i} \right);$$

end

$$z_i^{k+1} = \begin{cases} \frac{1}{L_i \|a_i\|^2 N} \nabla f_i(a_i^T \mathbf{prox}_{L^{-1}g}(x^{k-d_k})) & \text{if } i = i_k; \\ z_i^k & \text{otherwise.} \end{cases}$$

$$y_{N+1}^k = \left(I - \mathbf{prox}_{L^{-1}g}\right)(x^{k-d_k});$$

- Monotropic Programming?

$$\underset{x_j \in \mathcal{H}_j}{\text{minimize}} \sum_{j=1}^{M} g_j(x_j) + f(x_1, \ldots, x_M);$$

$$\text{subject to: } \sum_{j=1}^{M} A_j x_j = b$$

- Monotropic Programming?

$$\underset{x_j \in \mathcal{H}_j}{\text{minimize}} \sum_{j=1}^{M} g_j(x_j) + f(x_1, \ldots, x_M);$$

$$\text{subject to: } \sum_{j=1}^{M} A_j x_j = b$$

- Operator $S : \prod_{j=1}^{M+1} \mathcal{H}_j \to \prod_{j=1}^{M+1} \mathcal{H}_j$:

$$(S(x))_{M+1} := -\gamma_{M+1} \left(\sum_{l=1}^{M} A_l x_l - b \right);$$

$$(S(x))_j$$

$$:= x_j - \mathbf{prox}_{\gamma_j g_j} \left(x_j - \gamma_j A_j^* \left(x_{M+1} + 2\gamma_{M+1} \left(\sum_{l=1}^{M} A_l x_l - b \right) \right) - \gamma_j \nabla_j f(x) \right).$$

- $x^* \in \mathrm{zer}(S) \implies (x_1^*, \ldots, x_m^*)$ solves the monotropic programming problem. (Why?)

- **TropicSMART:**

Sample coordinate $j_k \in \{1, \ldots, M+1\}$ uniformly and set $\mathsf{S}_k = \{j_k\}$.

$$\overline{x}_{M+1}^{k+1} = x_{M+1}^k + \gamma_{M+1}\left(\sum_{l=1}^{M} A_l x_l^k - b\right);$$

$$\overline{x}_j^{k+1} = \mathbf{prox}_{\gamma_j g_j}\left(x_j^k - \gamma_j A_j^*(2\overline{x}_{M+1}^{k+1} - x_{M+1}^k) - \gamma_j \nabla_j f(x^k)\right);$$

$$(\forall j \in \mathsf{S}_k) \qquad x_j^{k+1} = x_j^k - \lambda\left(x_j^k - \overline{x}_j^{k+1}\right);$$

$$(\forall j \notin \mathsf{S}_k) \qquad x_j^{k+1} = x_j^k.$$

- More special cases in the paper.
- What about theory?

Theorem (SMART converges)

1. *The sequence* $\{x^k\}_{k\in\mathbb{N}}$ *weakly converges to a root of* S*.*
2. *If* S *is essentially strongly quasi monotone (ESQM)*

$$(\exists \mu > 0) : (\forall x \in \mathcal{H}) \quad \langle S(x), x - P_{\mathrm{zer}(S)}(x)\rangle \geq \mu \|x - P_{\mathrm{zer}(S)}(x)\|^2,$$

then $\{x^k\}_{k\in\mathbb{N}}$ *linearly converges to a root of* S*.*

- Examples of ESQM include $S(x) = \sum_i a_i \nabla f_i(a_i^T x)$ if each f_i is strongly convex.

- Proof is not difficult, but kind of long.

  1. Construct a supermartingale sequence

     $$\mathbb{E}[X_{k+1}|\mathcal{F}_k] + Y_k \leq X_k$$

     where

     $X_k =$ (distance to solution)^2 + (asynchrony residual) + (dual variables residual)

     $Y_k =$ (Residuals that force convergence if they are 0).

  2. Then through a series of magical steps, show that the sequence weakly converges.

- Linear convergence is somewhat more difficult to show.

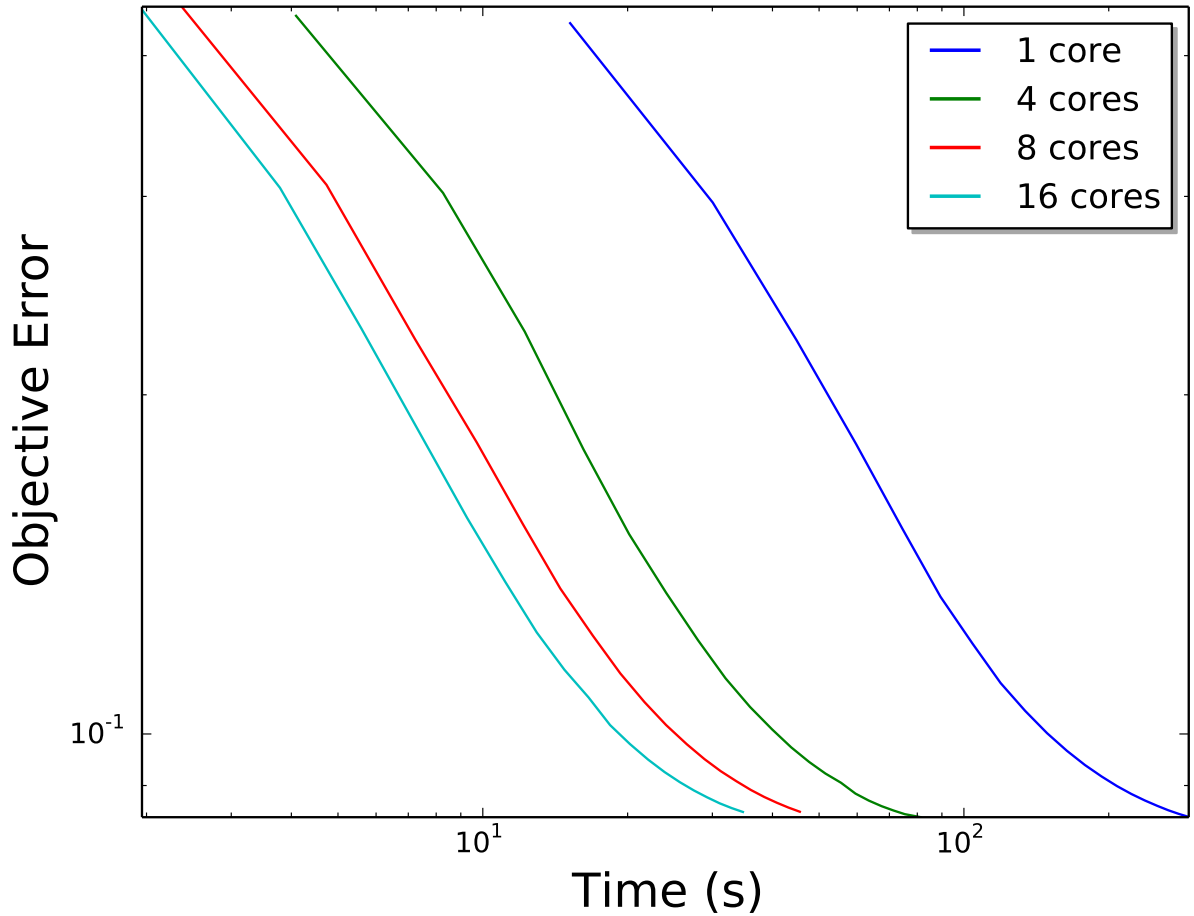

Figure: ℓ_2-regularized Logistic regression with $N = 1000$, $m = 10000$, condition number $= 10$, matrix A random Gaussian, vector b uniformly distributed.

- **A lot left to do.**

- Nonconvex case
  - (asynchronous matrix factorization algorithms soon)

- Do more numerical experiments
  - Brent Edmunds at UCLA making program that takes operators as input and runs SMART to find roots.

- Characterize sublinear convergence rates.

- Make more operators $\implies$ more algorithms.

Thanks!

- Paper available here: `http://arxiv.org/abs/1601.00698`
- This material is based upon work supported by the National Science Foundation under Award No. 1502405.