Minimizing Finite Sums with the Stochastic Average Gradient Algorithm

Mark Schmidt

Joint work with Nicolas Le Roux and Francis Bach

Simon Fraser University
Large-scale machine learning: large $N$, large $P$

- $N$: number of observations (inputs)
- $P$: dimension of each observation

Regularized empirical risk minimization: find $x^*$ solution of

$$\min_{x \in \mathbb{R}^P} \frac{1}{N} \sum_{i=1}^{N} \ell(x^T a_i) + \lambda r(x)$$

Data fitting term + regularizer

Applications to any data-oriented field: Vision, bioinformatics, speech, natural language, web.

Main practical challenges: Choosing regularizer $r(x)$ and data-fitting term $f(x)$. Designing/learning good features $a_i$. Efficiently solving the problem when $N$ or $P$ are very large.
**Large-scale machine learning**: large $N$, large $P$
- $N$: number of observations (inputs)
- $P$: dimension of each observation

**Regularized empirical risk minimization**: find $x^*$ solution of

$$\min_{x \in \mathbb{R}^P} \frac{1}{N} \sum_{i=1}^N \ell(x^T a_i) + \lambda r(x)$$

data fitting term + regularizer

**Applications to any data-oriented field**:
- Vision, bioinformatics, speech, natural language, web.

**Main practical challenges**:
- Choosing regularizer $r$ and data-fitting term $f$.
- Designing/learning good features $a_i$.
- Efficiently solving the problem when $N$ or $P$ are very large.
We want to minimize the sum of a finite set of smooth functions:

\[
\min_{x \in \mathbb{R}^P} g(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x).
\]
We want to minimize the sum of a finite set of smooth functions:

$$\min_{x \in \mathbb{R}^P} g(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x).$$

We are interested in cases where $N$ is very large.

We will focus on strongly-convex functions $g$. 
We want to minimize the sum of a finite set of smooth functions:

$$\min_{x \in \mathbb{R}^P} g(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x).$$

We are interested in cases where $N$ is very large.

We will focus on strongly-convex functions $g$.

Simplest example is $\ell_2$-regularized least-squares,

$$f_i(x) := (a_i^T x - b_i)^2 + \frac{\lambda}{2} \|x\|^2.$$

Other examples include any $\ell_2$-regularized convex loss:

- logistic regression, Huber regression, smooth SVMs, CRFs, etc.
We consider minimizing \( g(x) = \frac{1}{n} \sum_{i=1}^{N} f_i(x) \).
We consider minimizing $g(x) = \frac{1}{n} \sum_{i=1}^{N} f_i(x)$.

**Deterministic gradient method [Cauchy, 1847]:**

$$x_{t+1} = x_t - \alpha_t g'(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^{N} f_i'(x_t).$$

- Linear convergence rate: $O(\rho^t)$.
- Iteration cost is linear in $N$.
- Fancier methods exist, but still in $O(N)$.
We consider minimizing \( g(x) = \frac{1}{n} \sum_{i=1}^{N} f_i(x) \).

**Deterministic** gradient method [Cauchy, 1847]:

\[
x_{t+1} = x_t - \alpha_t g'(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^{N} f'_i(x_t).
\]

- **Linear** convergence rate: \( O(\rho^t) \).
- Iteration cost is linear in \( N \).
- Fancier methods exist, but still in \( O(N) \)

**Stochastic** gradient method [Robbins & Monro, 1951]:

- Random selection of \( i(t) \) from \( \{1, 2, \ldots, N\} \),

\[
x_{t+1} = x_t - \alpha_t f'_{i(t)}(x_t).
\]

- Iteration cost is independent of \( N \).
- **Sublinear** convergence rate: \( O(1/t) \).
We consider minimizing $g(x) = \frac{1}{n} \sum_{i=1}^{N} f_i(x)$.

**Deterministic gradient method [Cauchy, 1847]:**

**Stochastic gradient method [Robbins & Monro, 1951]:**
Motivation for New Methods

- **FG method** has $O(N)$ cost with linear rate.
- **SG method** has $O(1)$ cost with sublinear rate.

Stochastic vs. deterministic methods

- Goal = best of both worlds: linear rate with $O(1)$ iteration cost.

<table>
<thead>
<tr>
<th>stochastic</th>
<th>deterministic</th>
</tr>
</thead>
<tbody>
<tr>
<td>log(excess cost)</td>
<td>time</td>
</tr>
</tbody>
</table>

**Mark Schmidt**

**Minimizing Finite Sums with the SAG Algorithm**
Motivation for New Methods

- FG method has $O(N)$ cost with linear rate.
- SG method has $O(1)$ cost with sublinear rate.

Goal is linear rate and $O(1)$ cost.
A variety of methods have been proposed to speed up SG methods:

- **Step-size strategies, momentum, gradient/iterate averaging**

- **Stochastic version of accelerated and Newton-like methods**
  - Bordes et al. (2009), Sunehag et al. (2009), Ghadimi and Lan (2010), Martens (2010), Xiao (2010), Duchi et al. (2011)
A variety of methods have been proposed to speed up SG methods:

- **Step-size strategies, momentum, gradient/iterate averaging**

- **Stochastic version of accelerated and Newton-like methods**
  - Bordes et al. (2009), Sunehag et al. (2009), Ghadimi and Lan (2010), Martens (2010), Xiao (2010), Duchi et al. (2011)

- **None of these methods improve on the $O(1/t)$ rate**
Prior Work on Speeding up SG Methods

Existing linear convergence results:

- **Constant step-size SG, accelerated SG**
  - Linear convergence but only up to a fixed tolerance

- **Hybrid methods, incremental average gradient**
  - Linear rate but iterations make full passes through the data
Existing linear convergence results:

- **Constant step-size SG, accelerated SG**
  - Linear convergence but only up to a fixed tolerance

- **Hybrid methods, incremental average gradient**
  - Linear rate but iterations make full passes through the data

- **Special Problems Classes**
  - Collins et al. (2008), Strohmer & Vershynin (2009), Schmidt and Le Roux (2012), Shalev-Shwartz and Zhang (2012)
  - Linear rate but limited choice for the $f_i$'s
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?

YES!
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?

YES! The stochastic average gradient (SAG) algorithm:

- Randomly select $i(t)$ from $\{1, 2, \ldots, n\}$ and compute $f'_{i(t)}(x^t)$.

$$x^{t+1} = x^t - \alpha^t \frac{N}{N} \sum_{i=1}^{N} f'_i(x^t)$$

Memory: $y_{ti} = f'_{i(t)}(x^k)$ from the last $t$ where $i(t)$ was selected.

Assumes that gradients of other examples don’t change.

This assumption becomes accurate as $\|x^{t+1} - x^t\| \to 0$. 

Stochastic variant of increment average gradient (IAG). 

[Blatt et al. 2007]
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?

YES! The stochastic average gradient (SAG) algorithm:

- Randomly select $i(t)$ from $\{1, 2, \ldots, n\}$ and compute $f'_{i(t)}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^{N} f'_i(x^t)$$
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?

YES! The stochastic average gradient (SAG) algorithm:

- Randomly select $i(t)$ from $\{1, 2, \ldots, n\}$ and compute $f'_{i(t)}(x^t)$.

\[
x^{t+1} = x^t - \alpha^t \frac{N}{\sum_{i=1}^{N} y_i^t}
\]

- Memory: $y_i^t = f'_i(x^k)$ from the last $t$ where $i$ was selected.
Is it possible to have a general linearly convergent algorithm with iteration cost independent of $N$?

YES! The stochastic average gradient (SAG) algorithm:

- Randomly select $i(t)$ from \( \{1, 2, \ldots, n\} \) and compute $f'_{i(t)}(x^t)$.

$$
x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^{N} y^t_i
$$

- **Memory**: $y^t_i = f'_{i}(x^k)$ from the last $t$ where $i$ was selected.
- Assumes that gradients of other examples don’t change.
- This assumption becomes accurate as $\|x^{t+1} - x^t\| \to 0$.
- **Stochastic** variant of increment average gradient (IAG).

[Blatt et al. 2007]
Assume only that:

- $f_i$ is convex, $f'_i$ is $L$–continuous, $g$ is $\mu$-strongly convex.
Convergence Rate of SAG

Assume only that:

- $f_i$ is convex, $f'_i$ is $L$–continuous, $g$ is $\mu$-strongly convex.

**Theorem.** With $\alpha_t = \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] \leq \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8N}\right\}\right)^t C.$$

- Linear convergence with iteration cost independent of $N$.
- A linear rate is also achieved for any $\alpha_t \leq \frac{1}{16L}$.
Convergence Rate of SAG

Assume only that:
- $f_i$ is convex, $f'_i$ is $L$–continuous, $g$ is $\mu$-strongly convex.

**Theorem.** With $\alpha_t = \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] \leq \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8N}\right\}\right)^t C.$$ 

- **Linear convergence with iteration cost independent of $N$.**
- A linear rate is also achieved for any $\alpha_t \leq \frac{1}{16L}$.
  - Well-conditioned problems: *constant non-trivial reduction per pass*:
    $$\left(1 - \frac{1}{8N}\right)^N \leq \exp\left(-\frac{1}{8}\right) = 0.8825.$$  
  - Badly-conditioned problems, *almost same as deterministic method.* (gradient method has rate $(1 - \frac{\mu}{L})^{2t}$ if $\alpha_t = \frac{1}{L}$, but $N$ times slower)
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:
Assume that \( N = 700000, L = 0.25, \mu = 1/N \):
- Gradient method has rate \( \left( \frac{L-\mu}{L+\mu} \right)^2 = 0.99998 \).
- Accelerated gradient method has rate \( 1 - \sqrt{\mu L} = 0.99761 \).
- SAG (\( N \) iterations) has rate \( 1 - \min\{\mu 16, 1/8 N\} = 0.88250 \).
- Fastest possible first-order method:
  \[ \left( \sqrt{L} - \sqrt{\mu \sqrt{L} + \mu} \right)^2 = 0.99048 \]
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate $\left(\frac{L - \mu}{L + \mu}\right)^2 = 0.99998$.
- Accelerated gradient method has rate $\left(1 - \sqrt{\frac{\mu}{L}}\right) = 0.99761$. 

SAG ($N$ iterations) has rate $\left(1 - \min\{\frac{\mu}{16L}, \frac{1}{8N}\}\right)^N = 0.88250$. 

Fastest possible first-order method: $\left(\sqrt{L} - \sqrt{\mu} \sqrt{L}\right)^2 = 0.99048$. 

SAG beats two lower bounds: Stochastic gradient bound (linear vs. sub-linear). 
Full gradient bound (for typical $L$, $\mu$, and $N$). 

Number of $f'_{i}$ evaluations to reach $\epsilon$:
- Gradient: $O\left(\frac{NL\mu \log(1/\epsilon)}{\epsilon}\right)$. 
- Accelerated: $O\left(\frac{N\sqrt{L}\mu \log(1/\epsilon)}{\epsilon}\right)$. 
- SAG: $O\left(\max\{N, \frac{L\mu}{16}\} \log \frac{1}{\epsilon}\right)$. 

Mark Schmidt

Minimzing Finite Sums with the SAG Algorithm
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate $\left( \frac{L - \mu}{L + \mu} \right)^2 = 0.99998$.
- Accelerated gradient method has rate $\left(1 - \sqrt{\frac{\mu}{L}}\right) = 0.99761$.
- SAG ($N$ iterations) has rate $\left(1 - \min\left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\}\right)^N = 0.88250$. 

Mark Schmidt

Minimizing Finite Sums with the SAG Algorithm
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate \((\frac{L - \mu}{L + \mu})^2 = 0.99998\).
- Accelerated gradient method has rate \((1 - \sqrt{\frac{\mu}{L}}) = 0.99761\).
- SAG (\(N\) iterations) has rate \((1 - \min \left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\})^N = 0.88250\).
- Fastest possible first-order method: \(\left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2 = 0.99048\).
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate $(\frac{L-\mu}{L+\mu})^2 = 0.99998$.
- Accelerated gradient method has rate $(1 - \sqrt{\frac{\mu}{L}}) = 0.99761$.
- SAG ($N$ iterations) has rate $(1 - \min \{ \frac{\mu}{16L}, \frac{1}{8N} \})^N = 0.88250$.
- Fastest possible first-order method: $(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}})^2 = 0.99048$.

SAG beats two lower bounds:

- Stochastic gradient bound (linear vs. sub-linear).
- Full gradient bound (for typical $L$, $\mu$, and $N$).
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate $\left( \frac{L-\mu}{L+\mu} \right)^2 = 0.99998$.
- Accelerated gradient method has rate $\left( 1 - \sqrt{\frac{\mu}{L}} \right) = 0.99761$.
- SAG ($N$ iterations) has rate $\left( 1 - \min \left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\} \right)^N = 0.88250$.
- *Fastest possible* first-order method: $\left( \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}} \right)^2 = 0.99048$.

SAG beats two lower bounds:

- Stochastic gradient bound (linear vs. sub-linear).
- Full gradient bound (for typical $L$, $\mu$, and $N$).

Number of $f'_i$ evaluations to reach $\epsilon$:

- Gradient: $O(N \frac{L}{\mu} \log(1/\epsilon))$. 
Rate of Convergence Comparison

- Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:
  - Gradient method has rate $\left(\frac{L - \mu}{L + \mu}\right)^2 = 0.99998$.
  - Accelerated gradient method has rate $(1 - \sqrt{\frac{\mu}{L}}) = 0.99761$.
  - SAG ($N$ iterations) has rate $(1 - \min\left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\})^N = 0.88250$.
  - *Fastest possible* first-order method: $\left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2 = 0.99048$.

- **SAG beats two lower bounds:**
  - Stochastic gradient bound (linear vs. sub-linear).
  - Full gradient bound (for typical $L$, $\mu$, and $N$).

- **Number of $f_i'$ evaluations to reach $\epsilon$:**
  - Gradient: $O(N \frac{L}{\mu} \log(1/\epsilon))$.
  - Accelerated: $O(N \sqrt{\frac{L}{\mu}} \log(1/\epsilon))$. 
Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:

- Gradient method has rate $\left( \frac{L-\mu}{L+\mu} \right)^2 = 0.99998$.
- Accelerated gradient method has rate $(1 - \sqrt{\frac{\mu}{L}}) = 0.99761$.
- SAG ($N$ iterations) has rate $\left(1 - \min \left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\} \right)^N = 0.88250$.
- Fastest possible first-order method: $\left( \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \right)^2 = 0.99048$.

- SAG beats two lower bounds:
  - Stochastic gradient bound (linear vs. sub-linear).
  - Full gradient bound (for typical $L$, $\mu$, and $N$).

- Number of $f_i'$ evaluations to reach $\epsilon$:
  - Gradient: $O(N \frac{L}{\mu} \log(1/\epsilon))$.
  - Accelerated: $O(N \sqrt{\frac{L}{\mu}} \log(1/\epsilon))$.
  - SAG: $O(\max\{N, \frac{L}{\mu}\} \log(1/\epsilon))$. 
Theorem. With \( \alpha_t = \frac{1}{16L} \) the SAG iterations satisfy

\[
\mathbb{E}[g(x^t) - g(x^*)] \leq \left( 1 - \min \left\{ \frac{\mu}{16L}, \frac{1}{8N} \right\} \right)^t C,
\]

where if we initialize with \( y_i^0 = 0 \) we have

\[
C = [g(x^0) - g(x^*)] + \frac{4L}{N} \| x^0 - x^* \|^2 + \frac{\sigma^2}{16L},
\]

and if we initialize with \( y_i^0 = f'_i(x^0) - g'(x^0) \) we have

\[
C = \frac{3}{2} [g(x^0) - g(x^*)] + \frac{4L}{N} \| x^0 - x^* \|^2.
\]

- If we initialize with \( N \) iterations of SG, \([g(x^0) - g(x^*)]\) and \( \| x^0 - x^* \|^2 \) are in \( O(1/N) \) so \( C = O(1/N) \).
Convergence Rate in Convex Case

Assume only that:

- $f_i$ is convex, $f'_i$ is $L$–continuous, some $x^*$ exists.
Assume only that:

- $f_i$ is convex, $f_i'$ is $L$–continuous, some $x^*$ exists.

**Theorem.** With $\alpha_t \leq \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] = O(1/N)$$

- Faster than SG lower bound of $O(1/\sqrt{N})$. 

---

Mark Schmidt

Minimizing Finite Sums with the SAG Algorithm
Convergence Rate in Convex Case

Assume only that:

- $f_i$ is convex, $f_i'$ is $L$–continuous, some $x^*$ exists.

**Theorem.** With $\alpha_t \leq \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] = O(1/N)$$

- Faster than SG lower bound of $O(1/\sqrt{N})$.
- Same algorithm and step-size as strongly-convex case:
  - Algorithm is adaptive to strong-convexity.
  - Faster convergence rate if $\mu$ is locally bigger around $x^*$. 

Mark Schmidt  
Minimizing Finite Sums with the SAG Algorithm
Convergence Rate in Convex Case

Assume only that:

- $f_i$ is convex, $f'_i$ is $L$–continuous, some $x^*$ exists.

**Theorem.** With $\alpha_t \leq \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] = O(1/N)$$

- Faster than SG lower bound of $O(1/\sqrt{N})$.
- Same algorithm and step-size as strongly-convex case:
  - Algorithm is adaptive to strong-convexity.
  - Faster convergence rate if $\mu$ is locally bigger around $x^*$.
- Same algorithm could be used in non-convex case.
Assume only that:

- $f_i$ is convex, $f'_i$ is $L$-continuous, some $x^*$ exists.

**Theorem.** With $\alpha_t \leq \frac{1}{16L}$ the SAG iterations satisfy

$$\mathbb{E}[g(x^t) - g(x^*)] = O(1/N)$$

- Faster than SG lower bound of $O(1/\sqrt{N})$.
- Same algorithm and step-size as strongly-convex case:
  - Algorithm is adaptive to strong-convexity.
  - Faster convergence rate if $\mu$ is locally bigger around $x^*$.
- Same algorithm could be used in non-convex case.
- Contrast with SDCA:
  - Requires explicit strongly-convex regularizer.
  - Not adaptive to $\mu$, does not allow $\mu = 0$. 
Comparing FG and SG Methods

- quantum \((n = 50000, p = 78)\) and rcv1 \((n = 697641, p = 47236)\)

- Comparison of competitive deterministic and stochastic methods.
SAG Compared to FG and SG Methods

- quantum \((n = 50000, p = 78)\) and rcv1 \((n = 697641, p = 47236)\)

- SAG starts fast and stays fast.
quantum \((n = 50000, p = 78)\) and rcv1 \((n = 697641, p = 47236)\)

- PCD/DCA are similar on some problems, much worse on others.
while(1)
  Sample $i$ from $\{1, 2, \ldots, N\}$.
  Compute $f'_i(x)$.
  $d = d - y_i + f'_i(x)$.
  $y_i = f'_i(x)$.
  $x = x - \frac{\alpha}{N} d.$
while(1)

- Sample $i$ from $\{1, 2, \ldots, N\}$.
- Compute $f'_i(x)$.
- $d = d - y_i + f'_i(x)$.
- $y_i = f'_i(x)$.
- $x = x - \frac{\alpha}{N} d$.

Issues:

- Should we normalize by $N$?
SAG Implementation Issues

while(1)

- Sample $i$ from $\{1, 2, \ldots, N\}$.
- Compute $f_i'(x)$.
- $d = d - y_i + f_i'(x)$.
- $y_i = f_i'(x)$.
- $x = x - \frac{\alpha}{N} d$.

Issues:

- Should we normalize by $N$?
- Can we reduce the memory?
while(1)

- Sample $i$ from $\{1, 2, \ldots, N\}$.
- Compute $f'_i(x)$.
- $d = d - y_i + f'_i(x)$.
- $y_i = f'_i(x)$.
- $x = x - \frac{\alpha}{N} d$.

Issues:

- Should we normalize by $N$?
- Can we reduce the memory?
- Can we handle sparse data?
while(1)

- Sample $i$ from $\{1, 2, \ldots, N\}$.
- Compute $f'_i(x)$.
- $d = d - y_i + f'_i(x)$.
- $y_i = f'_i(x)$.
- $x = x - \frac{\alpha}{N} d$.

Issues:

- Should we normalize by $N$?
- Can we reduce the memory?
- Can we handle sparse data?
- How should we set the step size?
SAG Implementation Issues

- while(1)
  - Sample \( i \) from \{1, 2, \ldots, N\}.
  - Compute \( f'_i(x) \).
  - \( d = d - y_i + f'_i(x) \).
  - \( y_i = f'_i(x) \).
  - \( x = x - \frac{\alpha}{N} d \).

- Issues:
  - Should we normalize by \( N \)?
  - Can we reduce the memory?
  - Can we handle sparse data?
  - How should we set the step size?
  - When should we stop?
while(1)

- Sample $i$ from $\{1, 2, \ldots, N\}$.
- Compute $f'_i(x)$.
- $d = d - y_i + f'_i(x)$.
- $y_i = f'_i(x)$.
- $x = x - \frac{\alpha}{N} d$.

Issues:

- Should we normalize by $N$?
- Can we reduce the memory?
- Can we handle sparse data?
- How should we set the step size?
- When should we stop?
- Can we use mini-batches?
SAG Implementation Issues

while(1)
  Sample $i$ from $\{1, 2, \ldots, N\}$.
  Compute $f'_i(x)$.
  $d = d - y_i + f'_i(x)$.
  $y_i = f'_i(x)$.
  $x = x - \frac{\alpha}{N} d$.

Issues:
  Should we normalize by $N$?
  Can we reduce the memory?
  Can we handle sparse data?
  How should we set the step size?
  When should we stop?
  Can we use mini-batches?
  Should we shuffle the data?
Implementation Issues: Normalization

- Should we normalize by $N$ in the early iterations?
- The parameter update:
  - $x = x - \frac{\alpha}{N} d$. 
Implementation Issues: Normalization

Should we normalize by $N$ in the early iterations?

The parameter update:

$$x = x - \frac{\alpha}{M} d.$$  

We normalize by number of examples seen ($M$).

Better performance on early iterations.
Can we reduce the memory?

The memory update for $f_i(a_i^T x)$:

- Compute $f'_i(a_i^T x)$.
- $d = d - (y_i - f'_i(a_i^T x))$.
- $y_i = f'_i(a_i^T x)$. 

Only store the scalars $f'_i(\delta)$. Reduces the memory from $O(NP)$ to $O(N)$. 

Mark Schmidt
Minimizing Finite Sums with the SAG Algorithm
Can we reduce the memory?

The memory update for $f_i(a_i^T x)$:

- Compute $f'_i(\delta)$, with $\delta = a_i^T x$.
- $d = d - a_i(y_i - f'(\delta))$.
- $y_i = f'_i(\delta)$.

Only store the scalars $f'_i(\delta)$.

Reduces the memory from $O(NP)$ to $O(N)$. 
Can we handle **sparse** data?

The parameter update for each variable $j$:

\[ x_j = x_j - \frac{\alpha}{M} d_j. \]
Can we handle sparse data?

The parameter update for each variable $j$:

- $x_j = x_j - \frac{k \alpha}{M} d_j$.

For sparse data, $d_j$ is typically constant.

Apply previous $k$ updates when it changes.
Can we handle sparse data?

The parameter update for each variable $j$:

- $x_j = x_j - \frac{k\alpha}{M} d_j$.

For sparse data, $d_j$ is typically constant.

Apply previous $k$ updates when it changes.

Reduces the iteration cost from $O(P)$ to $O(\|f'_i(x)\|_0)$.

Standard tricks allow $\ell_2$-regularization and $\ell_1$-regularization.
How should we set the step size?

\[ \alpha = \frac{1}{16L} \]

Practice:
\[ \alpha = \frac{1}{L} \]

What if \( L \) is unknown or smaller near \( x^* \)?

Start with a small \( L \).
Increase \( L \) until we satisfy:
\[ f_i(x + \gamma L f'_i(x)) \leq f'_i(x) - \frac{1}{2L} \| f'_i(x) \|^2 \]
(assuming \( \| f'_i(x) \|^2 \geq \epsilon \))
Decrease \( L \) between iterations.

For \( f'_i(a^T x) \), this line-search is \( O(1) \) in \( N \) and \( P \):
How should we set the step size?

- Theory: $\alpha = 1/16L$.
- Practice: $\alpha = 1/L$.

What if $L$ is unknown or smaller near $x^*$?

Start with a small $L$. Increase $L$ until we satisfy:

$$f_i(x + \frac{1}{L} f'_i(x)) \leq f'_i(x) - \frac{1}{2L} \|f'_i(x)\|^2$$

(assuming $\|f'_i(x)\|^2 \geq \epsilon$)

Decrease $L$ between iterations.

For $f'_i(a^T x)$, this line-search is $O(1)$ in $N$ and $P$: $f'_i(a^T x - \frac{1}{L} f'_i(\delta)) \leq (a^T \|a\|_2)$.
Implementation Issues: Step-Size

- How should we set the step size?
  - Theory: \( \alpha = 1/16L \).
  - Practice: \( \alpha = 1/L \).
- What if \( L \) is unknown or smaller near \( x^* \)?
Implementation Issues: Step-Size

- How should we set the step size?
  - Theory: $\alpha = 1/16L$.
  - Practice: $\alpha = 1/L$.

- What if $L$ is unknown or smaller near $x^*$?
  - Start with a small $L$.
  - Increase $L$ until we satisfy:
    \[
    f_i(x^+ - \frac{1}{L} f'_i(x)) \leq f'_i(x) - \frac{1}{2L} \| f'_i(x) \|^2.
    \]
    (assuming $\| f'_i(x) \|^2 \geq \epsilon$)
  - Decrease $L$ between iterations.
How should we set the step size?
- **Theory:** $\alpha = 1/16L$.
- **Practice:** $\alpha = 1/L$.

What if $L$ is unknown or smaller near $x^*$?
- Start with a small $L$.
- **Increase $L$** until we satisfy:

$$f_i(x^+ - \frac{1}{L} f'_i(x)) \leq f'_i(x) - \frac{1}{2L} \|f'_i(x)\|^2.$$

(assuming $\|f'_i(x)\|^2 \geq \epsilon$)
- **Decrease $L$** between iterations.

For $f'_i(a_i^T x)$, this line-search is $O(1)$ in $N$ and $P$:

$$f'_i(a_i^T x - \frac{f'(\delta)}{L} \|a_i\|^2).$$
When should we stop?
When should we stop?
Normally we check the size of $\|f'(x)\|$. 
When should we stop?

Normally we check the size of $\|f'(x)\|$.

And SAG has $y_i \rightarrow f'_i(x)$. 
When should we stop?

Normally we check the size of $\|f'(x)\|$.

And SAG has $y_i \rightarrow f'_i(x)$.

We can check the size of $\left\| \frac{1}{N} d \right\| = \left\| \frac{1}{N} \sum_{i=1}^{N} y_i \right\| \rightarrow \|f'(x)\|$
Can we use mini-batches?

Yes, define each $f_i$ to include more than one example.

Reduces memory requirements.

Allows vectorization.

But must decrease $L$ for good performance: $L \leq \max_{i \in B} \{L_i\}$.

In practice, use the line-search on the batch to determine $L_B$. 
Can we use mini-batches?

- Yes, define each $f_i$ to include more than one example.
- Reduces memory requirements.
- Allows vectorization.
- But **must decrease** $L$ for good performance: $L_B \leq \max_{i \in B} \{L_i\}$. 
Can we use mini-batches?

- Yes, define each $f_i$ to include more than one example.
- Reduces memory requirements.
- Allows vectorization.
- But must decrease $L$ for good performance: $L_B \leq \max_{i \in B} \{L_i\}$.
- In practice, use the line-search on the batch to determine $L_B$. 
Implementation Issues: Non-Uniform Sampling

- Does re-shuffling and doing full passes work better?

\[ \text{Sample proportional to Lipschitz constants (skip the duplications).} \]

\[ \text{No re-weighting required. (just updates } y_i \text{ more often if } f'_i \text{ can change more quickly)} \]

Better performance using partially biased sampling.

Combine with the line-search for adaptive sampling. (see paper/code for details)

Mark Schmidt

Minimizing Finite Sums with the SAG Algorithm
Implementation Issues: Non-Uniform Sampling

- Does re-shuffling and doing full passes work better?
  - **NO!**

Now convergence rate depends on $L_{\text{mean}}$ instead of $L_{\text{max}}$.

Sample proportional to Lipschitz constants (skip the duplications). No re-weighting required. (just updates $y_i$ more often if $f_i'$ can change more quickly)

Better performance using partially biased sampling. Combine with the line-search for adaptive sampling. (see paper/code for details)
Implementation Issues: Non-Uniform Sampling

- Does re-shuffling and doing full passes work better?
  - NO!
  - Performance is intermediate between IAG and SAG.
Implementation Issues: Non-Uniform Sampling

- Does re-shuffling and doing full passes work better?
  - NO!
  - Performance is intermediate between IAG and SAG.
- Can non-uniform sampling help?

\[ \frac{1}{N} \sum_{i=1}^{N} f_i(x) = \frac{1}{\sum_{i=1}^{N} L_i} \sum_{i=1}^{N} \sum_{j=1}^{L_i} \left( f_i(x) \right) \]

Now convergence rate depends on \( L_{\text{mean}} \) instead of \( L_{\text{max}} \).

Sample proportional to Lipschitz constants (skip the duplications).

No re-weighting required.

(just updates \( y_i \) more often if \( f_i' \) can change more quickly)

Better performance using partially biased sampling.

Combine with the line-search for adaptive sampling.

(see paper/code for details)
Does re-shuffling and doing full passes work better?

NO!
Performance is intermediate between IAG and SAG.

Can non-uniform sampling help?

Duplicate examples proportional to their Lipschitz constants:

\[
\frac{1}{N} \sum_{i=1}^{N} f_i(x) = \frac{1}{\sum L_i} \sum_{i=1}^{N} \sum_{j=1}^{L_i} L_{\text{mean}} \frac{f_i(x)}{L_i}.
\]

Now convergence rate depends on \(L_{\text{mean}}\) instead of \(L_{\text{max}}\).
Implementation Issues: Non-Uniform Sampling

- Does re-shuffling and doing full passes work better?
  - NO!
  - Performance is intermediate between IAG and SAG.

- Can non-uniform sampling help?
  - Duplicate examples proportional to their Lipschitz constants:
    \[
    \frac{1}{N} \sum_{i=1}^{N} f_i(x) = \frac{1}{\sum L_i} \sum_{i=1}^{N} \sum_{j=1}^{L_i} L_{\text{mean}} \frac{f_i(x)}{L_i}.
    \]
  - Now convergence rate depends on $L_{\text{mean}}$ instead of $L_{\text{max}}$.
  - Sample proportional to Lipschitz constants (skip the duplications).
**Implementation Issues: Non-Uniform Sampling**

- Does **re-shuffling** and doing full passes work better?
  - **NO!**
  - Performance is intermediate between IAG and SAG.

- Can **non-uniform** sampling help?
  - Duplicate examples proportional to their Lipschitz constants:
    
    \[
    \frac{1}{N} \sum_{i=1}^{N} f_i(x) = \frac{1}{\sum_{i=1}^{N} L_i} \sum_{i=1}^{N} \sum_{j=1}^{L_i} L_{\text{mean}} \frac{f_i(x)}{L_i}.
    \]

  - Now convergence rate depends on \( L_{\text{mean}} \) instead of \( L_{\text{max}} \).
  - **Sample proportional to Lipschitz constants** (skip the duplications).
  - **No re-weighting required.**
    (just updates \( y_i \) more often if \( f_i' \) can change more quickly)
Does re-shuffling and doing full passes work better?
  - NO!
  - Performance is intermediate between IAG and SAG.

Can non-uniform sampling help?
  - Duplicate examples proportional to their Lipschitz constants:
    $$\frac{1}{N} \sum_{i=1}^{N} f_i(x) = \frac{1}{\sum L_i} \sum_{i=1}^{N} \sum_{j=1}^{L_i} L_{\text{mean}} \frac{f_i(x)}{L_i}.$$ 
  - Now convergence rate depends on $L_{\text{mean}}$ instead of $L_{\text{max}}$.
  - Sample proportional to Lipschitz constants (skip the duplications).
  - No re-weighting required.
    (just updates $y_i$ more often if $f_i'$ can change more quickly)
  - Better performance using partially biased sampling.
  - Combine with the line-search for adaptive sampling.
    (see paper/code for details)
SAG with Non-Uniform Sampling

- protein \((n = 145751, p = 74)\) and sido \((n = 12678, p = 4932)\)

- Datasets where SAG had the worst relative performance.

Mark Schmidt
Minimizing Finite Sums with the SAG Algorithm
SAG with Non-Uniform Sampling

- protein ($n = 145751$, $p = 74$) and sido ($n = 12678$, $p = 4932$)

- Lipschitz sampling helps a lot.
Conclusion and Discussion

- Faster theoretical convergence using only the ‘sum’ structure.
Conclusion and Discussion

- Faster theoretical convergence using only the ‘sum’ structure.
- Simple algorithm, *empirically better than theory predicts*. 

- Black-box stochastic gradient algorithm: Adaptivity to problem difficulty, line-search, termination criterion.
- Constrained and non-smooth problems: Proximal-gradient, ADMM.
- Memory-free methods: Similar performance, but requires two $f'$ evaluations per iteration.
- For the adventurous: Accelerated (seems to work, but requires a small step size). Newton-like (diagonal-scaling didn’t seem to help). Going asynchronous (algorithm seems to be robust to this). Simpler proof technique.
Conclusion and Discussion

- Faster theoretical convergence using only the ‘sum’ structure.
- Simple algorithm, empirically better than theory predicts.
- Black-box stochastic gradient algorithm:
  - Adaptivity to problem difficulty, line-search, termination criterion.
Faster theoretical convergence using only the ‘sum’ structure.
Simple algorithm, empirically better than theory predicts.
Black-box stochastic gradient algorithm:
  - Adaptivity to problem difficulty, line-search, termination criterion.
Constrained and non-smooth problems:
  - Proximal-gradient, ADMM.

Faster theoretical convergence using only the ‘sum’ structure.

Simple algorithm, empirically better than theory predicts.

Black-box stochastic gradient algorithm:
- Adaptivity to problem difficulty, line-search, termination criterion.

Constrained and non-smooth problems:
- Proximal-gradient, ADMM.

Memory-free methods:
- Similar performance, but requires two $f'_i$ evaluations per iteration.


[Mahdavi et al., 2013, Johnson and Zhang, 2013, Zhang et al., 2013, Konecny and Richtarik, 2013, Next talk, 2014]
Faster theoretical convergence using only the ‘sum’ structure.
Simple algorithm, empirically better than theory predicts.
Black-box stochastic gradient algorithm:
  - Adaptivity to problem difficulty, line-search, termination criterion.
Constrained and non-smooth problems:
  - Proximal-gradient, ADMM.
Memory-free methods:
  - Similar performance, but requires two $f'_i$ evaluations per iteration.
    - [Mahdavi et al., 2013, Johnson and Zhang, 2013, Zhang et al., 2013, Konecny and Richtarik, 2013, Next talk, 2014]]
For the adventurous:
  - Accelerated (seems to work, but requires a small step size).
  - Newton-like (diagonal-scaling didn’t seem to help).
  - Going asynchronous (algorithm seems to be robust to this).
  - Simpler proof technique.