COM S 578X: Optimization for Machine Learning
Project Report

Name: Jinu Susan Kabala   Email: jsusan@iastate.edu

December 11, 2018

1 Introduction

Suppose we want to minimize the sum of functions which are smooth and convex, where n is very large.

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

If we apply Gradient Descent(GD) to solve this problem, we have the following update in each k=1,2,...

$$x_{k+1} = x_k - s_k \sum_{i=1}^{n} \nabla f_i(x)$$

Under strong convexity, gradient descent has a linear convergence rate O(β^k). If the function f is convex and ∇f is L-Lipschitz continuous, then convergence rate is sub-linear O(1/k). The computational cost per update is O(n). When the side of n is very large, this is a high computational cost.

If we use a Stochastic Gradient Descent(SGD) method, then for k = 1,2..., the update is as follows.

$$x_{k+1} = x_k + s_k \nabla f_{i_k}(x_k)$$

where i_k is randomly selected index from 1...n. Here, $\tilde{g}_i(x_k) = \nabla f_{i_k}(x_k)$ is the noisy unbiased gradient at x_k. So $\mathbb{E}\nabla f_{i_k}(x_k) = \nabla f_i(x_k)$. The computation cost to compute the gradient is O(1), but the convergence rate is sub-linear O(1/k) which is slower than gradient descent if f is convex under diminishing step size. Under strong convexity and other conditions, SGD achieve $O(1/\sqrt{k})$ convergence in expectation. Eventhough the computation cost is much lesser than Gradient descent, SDG suffers slow convergence rate.

A variation of SGD is called mini-batch SGD, where a random subset of size b << n, $B_k \subset 1, 2, ...n$ is chosen, so that the update looks like following:

$$x_{k+1} = x_k + s_k \frac{1}{b} \sum_{i_k \in B_k} \nabla f_{i_k}(x_k)$$

Here the $\tilde{g}(x_k) = \frac{1}{b}\sum_{i_k \in B_k} \nabla f_{i_k}(x_k)$ is the noisy unbiased gradient at step k. Here, the computation cost to compute the gradient is O(b), but the convergence rate $O((1/\sqrt{bk}) + (1/k))$[1] which is faster that SGD.

Theorem 1. *[2] For SGD, under the following assumptions,*

- $f* = \inf_x f(x_k) > -\infty$, *with* $f(x^*) = f^*$
- $\mathbb{E}||\tilde{g}_k||^2 \leq G^2$, $\forall k$
- $\mathbb{E}||x_0 - x^*||_2^2 \leq R^2$, $\forall k$

and, step size $\{s_k\}_{k=1}^{\infty}$ *satisfies* $s_k > 0, \forall k$, $\sum_{k=1}^{\infty} s_k^2 = B < \infty$, $\sum_{k=1}^{\infty} s_k = \infty$, *we have:*

$$\lim_{k\to\infty} \mathbb{E}[f_{best}^{(k)}] = f^*$$

and

$$\lim_{k\to\infty} \mathbb{P}[|f_{best}^{(k)} = f^*| > \epsilon] = 0$$

$$\mathbb{P}[\lim_{k\to\infty} f_{best}^{(k)} = f^*] = 1$$

Also,

$$\min_{i=1...k} \{\mathbb{E}[f(x_k)] - f^*\} \leq \frac{R^2 + G^2 B}{2\sum_{i=1}^{k} s_i}$$

Hence, reducing the variance term, $\mathbb{E}||\tilde{g}_k||^2 \leq G^2$ can have an effect in the convergence rate. This report is a literature review of some variance reduction techniques used to reduce the variance of the noisy estimate of the gradient as a way to speed convergence so that as the numbe of steps, k increases, the variance diminishes to zero.

2 Variance Reduction

Let's look at the SGD update.

$$x_{k+1} = x_k - s_k \tilde{g}_i(x_k)$$

where $\tilde{g}_i(x_k)$ is the noisy (sub)gradient of $f_i(x_k)$.
The bias and variance of an estimator are as follows:

$$Bias(\tilde{g}_i(x_k)) = E[\tilde{g}_i(x_k) - f_i(x_k)] = E[\tilde{g}_i(x_k)] - f_i(x_k)$$

$$Var(\tilde{g}_i(x_k)) = E[\tilde{g}_i(x_k)]^2 - (E[\tilde{g}_i(x_k)])^2 \leq E[\tilde{g}_i(x_k)]^2$$

Here the intuition behind the variance reduction algorithm in introduced. Let $\tilde{g} = g^X - g^Y$ where g^X is an unbiased estimator of ∇f. $\mathbb{E}\tilde{g} = \nabla f$ if $\mathbb{E}g^Y \approx 0$, so that $\tilde{g}$ is an unbiased estimator of ∇f as well.

$$Var(\tilde{g}) = var(g^X - g^Y) = var(g^X) + var(g^Y) - 2cov(g^X, g^Y)$$

$Var(\tilde{g})$ can be reduced by having a high covariance between g^X and g^Y.

2.1 Stochastic Average Gradient(SAG)

Algorithm: (Schmidt, Roux, Bach[3])

- Initialize x_0.
- Maintain a table for storing gradient vector. For i=1...n, set $g_0 = x_0$
- while $||\nabla f(x_k)||_2^2 > 0$:
  - Pick i_k randomly from 1...n
  - Set $g_{i_k}(x_k) = \nabla f_{i_k}(x_k)$
  - Set all other $i \neq i_k$ to $g_i(x_k) = g_i(x_{k-1})$
  - $x_{k+1} = x_k - s_k \left(\frac{g_{i_k}(x_{k+1})}{n} - \frac{g_{i_k}(x_k)}{n} + \frac{1}{n}\sum_{i=1}^{n} g_i(x_k) \right)$

SAG gradient estimates are not unbiased, but reduces the variance. The following theorems[3] convey that under convexity and lipschitz continuity of ∇f_i, SAG converges linearly in expectation, O(β^k) under strong convexity. And, sublinear O(1/k) convergence under convexity and lipschitz continuity. This is similar to Gradient Descent. Since the gradients are stored, it has a higher memory cost that GD. If $x \in \mathbb{R}^p$, then SAG stores gradient for each training sample, incurring a memory cost of O(np).

Theorem 2. ***Schmidt, Roux, Bach[3]*** *Suppose $f(x) = \sum_{i=1}^n f_i(x)$ where each f_i is differentiable and the gradient, $\nabla f_i(x)$ is L-Lipschitz continuous. SAG with fixed step size $t = \frac{1}{16L}$ and the initialization*

$$g_i^{(0)} = \nabla f_i(x^{(0)}) - \nabla f(x^{(0)}), i = 1, ..., n$$

satisfies

$$\mathbb{E}[f(\bar{x}^{(k)})] - f^* \leq \frac{48n}{k}\left(f(x^{(0)} - f^*) + \frac{128L}{k}||x^{(0)} - x^*||_2^2\right)$$

where the expectation is taken over the random choice of index at each iteration.

Theorem 3. ***Schmidt, Roux, Bach [3]*** *Suppose $f(x) = \sum_{i=1}^n f_i(x)$ where each f_i is differentiable and the gradient, $\nabla f_i(x)$ is L-Lipschitz continuous and also μ strongly convex. SAG with fixed step size $t = \frac{1}{16L}$ and the initialization*

$$g_i^{(0)} = \nabla f_i(x^{(0)}) - \nabla f(x^{(0)}), i = 1, ..., n$$

satisfies

$$\mathbb{E}[f(\bar{x}^{(k)})] - f^* \leq \left(1 - min\left\{\frac{m}{16L}, \frac{1}{8n}\right\}\right)^k \left(\frac{3}{2}(f(x^{(0)}) - f^*) + \frac{4L}{n}||x^{(0)} - x^*||_2^2\right)$$

where the expectation is taken over the random choice of index at each iteration.

The proofs of these theorems are not trivial and thus skipped.

2.2 SAGA

Algorithm: (Defazio, Bach, and Lacoste-Julien[4])

- Initialize x_0.
- Maintain a table for storing gradient vector. For i=1...n, set $g_0 = x_0$
- while $||\nabla f(x_k)||_2^2 > 0$:
  - Pick i_k randomly from 1...n
  - Set $g_{i_k}(x_k) = \nabla f_{i_k}(x_k)$
  - Set all other $i \neq i_k$ to $g_i(x_k) = g_i(x_{k-1})$
  - $x_{k+1} = x_k - s_k\left(g_{i_k}(x_{k+1}) - g_{i_k}(x_k) + \frac{1}{n}\sum_{i=1}^n g_i(x_k)\right)$

The gradient estimate is unbiased unlike SAG. The convergence rates are similar to SAG, but has higher variance updates. SAGA can be applied to composite problems of the form

$$\min_x \frac{1}{n}\sum_{i=1}^n f_i(x) + h(x)$$

where f_i is smooth and convex but $h(x)$ is nonsmooth and convex. The memory cost is $O(np)$ like SAG.

2.3 Stochastic variance reduced gradient (SVRG)

Algorithm: (R. Johnson and T. Zhang [5]) Input update frequency, m and learning rate, s

- Initialize $\tilde{x}_0$.
- for $k = 1, 2, ...$ do:
  - Let $\tilde{x} = \tilde{x}_{k-1}$,
  - $\tilde{\mu} = \nabla P(\tilde{x}) = \frac{1}{n}\sum_{i=1}^n f_i(\tilde{x})$
  - Set $\tilde{x}_0 = \tilde{x}$.

- For t = 1...m:
  - Pick i_t randomly from {1,...,n}
  - $x_t = x_{t-1} - s_t \left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla P(\tilde{x}) \right)$
- Option I: $\tilde{x}_k = x_m$.
- Option II: $\tilde{x}_k = x_t$ where t is randomly chosen from {1,..., $m-1$}.

Convergence rates are similar to SAG and SAGA but does not store gradients of function in tables which reduces the memory cost. The computational cost for $\nabla P(\tilde{x})$ is O(n) and that of the update loop is O(m). For k iterations, the computational cost is O(k(m+n)). But the memory cost is still higher than SGD and GD. If $x \in \mathbb{R}^p$, then the memory cost is O(3p) which is much lesser than O(np) for SAG and SAGA.

The proofs for SVRG convergence is much simpler and similar to SGD[2]. The following is the proof for strongly convex function.

Theorem 4. *(R. Johnson and T. Zhang [5]) Consider SVRG with Option II. Assume that all the $f_i(x)$ are convex and smooth, $P(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ be strongly convex. Let $x^* = argmin_x P(x)$. Assume that m is sufficiently large so that:*

$$\alpha = \frac{1}{\gamma s(1-2Ls)m} + \frac{2Ls}{1-2Ls} < 1$$

Then we have a geometric convergence in expectation for SVRG. That is,

$$\mathbb{E}P(\tilde{x}_k) \leq \mathbb{E}P(\tilde{x}^*) + \alpha^k [P(\tilde{x}_0) - P(x^*)]$$

Proof. We know that, if $f \in \mathcal{F}_L^{1,1}$

$$f(y) \leq f(x) + \nabla f(x)^\top (y-x) + \frac{L}{2}||x-y||^2$$

If $f \in \mathcal{F}_L^{1,1}$ and $x_{k+1} = x_k - s\nabla f(x_k)$, then:

$$f(x_k - s\nabla f(x_k)) \leq f(x) - s\left(1 - \frac{Ls}{2}\right)||\nabla f(x_k)||^2$$

For any i= 1,...,n, we know that, Let

$$g_i(x) = f_i(x) - f_i(x^*) - \nabla f_i(x^*)^\top (x - x^*)$$

$$\nabla g_i(x) = \nabla f_i(x) - \nabla f_i(x^*)$$

$$\nabla P(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

$$\begin{aligned}
0 = g_i(x^*) &\leq min_s[g_i(x - s\nabla g_i(x))] \\
&\leq g_i(x) - s\left(1 - \frac{Ls}{2}\right)||\nabla g_i(x_k)||^2 \\
&= g_i(x) - s\left(1 - \frac{Ls}{2}\right)||\nabla g_i(x_k)||^2 \qquad (\text{ Setting } s = \frac{1}{L}) \\
&= g_i(x) - \frac{1}{2L}||\nabla g_i(x_k)||^2 \\
\implies g_i(x) &\leq -\frac{1}{2L}||\nabla g_i(x_k)||^2 \\
\implies ||\nabla g_i(x_k)||^2 &\leq 2Lg_i(x) \\
||\nabla f_i(x) - \nabla f_i(x^*||^2 &\leq 2L\left(f_i(x) - f_i(x^*) - \nabla f_i(x^*)^\top(x - x^*)\right) \\
\frac{1}{n}\sum_{i=1}^{n}||\nabla f_i(x) - \nabla f_i(x^*||^2 &\leq 2L\frac{1}{n}\sum_{i=1}^{n}\left(f_i(x) - f_i(x^*) - \nabla f_i(x^*)^\top(x - x^*)\right) \\
\mathbb{E}||\nabla f_i(x) - \nabla f_i(x^*)||^2 &\leq 2L\frac{1}{n}\left(f_i(x) - f_i(x^*) - \nabla f_i(x^*)^\top(x - x^*)\right) \\
&\leq 2L\left(P(x) - P(x^*)\right) ----(1)
\end{aligned}$$

An update happens such that $x_t = x_{t-1} - s\left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}\right)$ or $x_t = x_{t-1} - sv_t$ where

$$v_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}$$

is the approximate gradient of SVRG. Let's calculate the variance of the gradient.

$$\tilde{\mu} = \nabla P(\tilde{x})$$

$$\begin{aligned}
\mathbb{E}||v_t||^2 &= \mathbb{E}||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}||^2 \\
&= \mathbb{E}||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*) + \nabla f_{i_t}(x^*) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}||^2 \\
&\leq 2\mathbb{E}||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)||^2 + 2\mathbb{E}||\nabla f_{i_t}(x^*) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}||^2 \qquad (\text{ since } ||a + b||^2 \leq 2||a||^2 + 2||b||^2) \\
&= 2\mathbb{E}||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)||^2 + 2\mathbb{E}||\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*) - \tilde{\mu}||^2 \\
&\leq 2\mathbb{E}||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)||^2 + 2\mathbb{E}||\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)||^2 \\
&\leq 2(2LP(x_{t-1}) - P(x^*)) + 2(2LP(\tilde{x}) - P(x^*)) \qquad (\text{ from (1) }) \\
&= 4L(P(x_{t-1}) - P(x^*) + P(\tilde{x}) - P(x^*))
\end{aligned}$$

Lets look at the SVRG update $x_t = x_{t-1} - sv_t$

$$\begin{aligned}
\mathbb{E}[||x_t - x^*||^2|x_{t-1}] &= \mathbb{E}[||x_{t-1} - sv_t - x^*||^2|x_{t-1}] \\
&= ||x_{t-1} - x^*||^2 + s^2\mathbb{E}||v_t||^2 - 2s||x_{t-1} - x^*||\mathbb{E}v_t \\
&= ||x_{t-1} - x^*||^2 + s^2\mathbb{E}||v_t||^2 - 2s||x_{t-1} - x^*||\mathbb{E}[\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}] \\
&= ||x_{t-1} - x^*||^2 + s^2\mathbb{E}||v_t||^2 - 2s||x_{t-1} - x^*||[\nabla P(x_{t-1}) - \nabla P(\tilde{x}) + \nabla P(\tilde{x})] \\
&\leq ||x_{t-1} - x^*||^2 + 4Ls^2[(P(x_{t-1}) - P(x^*) + P(\tilde{x}) - P(x^*))] - 2s||x_{t-1} - x^*||\nabla P(x_{t-1}) \\
&\leq ||x_{t-1} - x^*||^2 + 4Ls^2[(P(x_{t-1}) - P(x^*) + P(\tilde{x}) - P(x^*))] - 2s||x_{t-1} - x^*||\nabla P(x_{t-1}) \\
&(\text{ since by convexity of P(x), } ||x_{t-1} - x^*||\nabla P(x_{t-1}) \leq P(x^*) - P(x_{t-1})) \\
&\leq ||x_{t-1} - x^*||^2 + 4Ls^2[(P(x_{t-1}) - P(x^*) + P(\tilde{x}) - P(x^*))] - 2s(P(x^*) - P(x_{t-1})) \\
&\leq ||x_{t-1} - x^*||^2 + -2s(1 - 2Ls)[P(x^*) - P(x_{t-1})] + 4Ls^2[P(\tilde{x}) - P(x^*))]
\end{aligned}$$

After all updates out of m are completed, $\tilde{x_k} = \tilde{x}_{k-1}$ Summing over t=1...m,

$$\sum_{i=1}^{m} \mathbb{E}[||x_t - x^*||^2]$$
$$\le \sum_{i=1}^{m} \{||x_{t-1} - x^*||^2 - 2s(1-2Ls)[P(x^*) - P(x_{t-1})] + 4Ls^2[P(\tilde{x}) - P(x^*))]\}$$
$$\mathbb{E}||x_m - x^*||^2 + 2s(1-2Ls)m\mathbb{E}[P(x^*) - P(x_k)] \le \mathbb{E}||\tilde{x} - x^*||^2 + 4Lms^2[P(\tilde{x}) - P(x^*))]$$
$$\le \frac{2}{\gamma}\mathbb{E}[P(\tilde{x}) - P(x^*)] + 4Lms^2[P(\tilde{x}) - P(x^*))]$$

where

$$2s(1-2Ls)m\mathbb{E}[P(\tilde{x}_k) - P(x^*)] \le \frac{2}{\gamma}\mathbb{E}[P(\tilde{x}) - P(x^*)] + 4Lms^2[P(\tilde{x}) - P(x^*))]$$
$$\mathbb{E}[P(\tilde{x}_k) - P(x^*)] \le \alpha\mathbb{E}[P(\tilde{x}) - P(x^*)]$$

where

$$\alpha = \frac{1}{\gamma s(1-2Ls)m} + \frac{2Ls}{1-2Ls}$$

For k iterations, we have,

$$\mathbb{E}[P(\tilde{x}_k) - P(x^*)] \le \alpha^k\mathbb{E}[P(\tilde{x}) - P(x^*)]$$

□

3 Evaluation

In order to evaluate the algorithms in this report, the optimization problem is:

$$\min_{\beta} \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(x_i^\top \beta - y_i)^2$$

This is a linear regression problem where the goal is to estimate β in order to minimized the loss function which is squared error loss. Here, (x_i, y_i), $i = 1, ..., n$ can be considered to be training samples. These samples are simulated as [6], where n is the sample size.

$$x_i \sim^{iid} Unif(-5, 5)$$
$$y_i \sim^{iid} N(x_i + 3, 1)$$

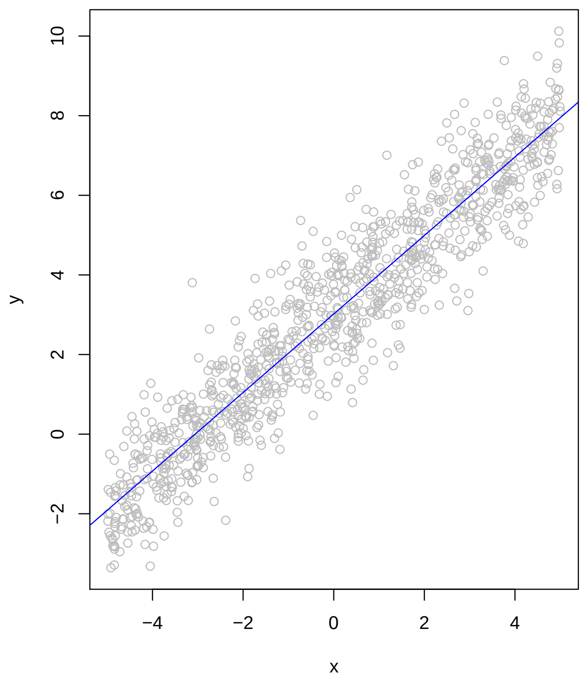
Experiment 1: Starting value $\beta_0 = (0,0)^\top$, n=1000. SAG and SAGA fixed step size of 0.01 and 0.03 respectively. GD, SGD have diminishing step size of $1/\sqrt{k}$

Table 1: Compare $\hat{\beta}$ estimates obtained by Linear Regression with Gradient Descent, SGD and SGD for mini-batches of size 10 and 100

method	time	steps	beta0	beta1
LR	0.00	0.00	3.03	1.00
GD	1.94	40.00	3.03	1.00
SGD-1	7.43	381.00	3.06	1.00
SGD-10	3.47	185.00	3.05	1.00
SGD-500	1.69	88.00	3.03	1.00

The $\hat{\beta}$ estimates are very close for all methods to the Linear Regression estimates. The number of steps taken to converge decreases with increase of size in the mini-batches. With this set-up, both SAG and SAGA failed to converge.

Figure 1: Linear Regression line fit to simulated data



The authors suggest using the estimate after n-steps of SGD as the starting value for SAG to make $f(x^{(0)} - f^*)$ small.. This suggestion did not work for me and it still diverged. The step size is tuned so that the step size is the maximum step size for which the algorithm does not diverge.

Experiment 2: Starting value $\beta_0 = (0, 0)^\top$, n=1000. SAG and SAGA fixed step size of 0.029 and 0.03 respectively. GD, SGD, SVRG have diminishing step size of $1/\sqrt{k}$

Table 2: Compare $\hat{\beta}$ estimates obtained by Linear Regression with Gradient Descent, SGD as well as SAG, SAGA and SVRG

method	time	steps	beta0	beta1
LR	0.00	0.00	3.03	1.00
GD	1.94	40.00	3.03	1.00
SGD-1	7.43	381.00	3.06	1.00
SAG	5.59	300.00	3.04	1.00
SAGA	2.41	134.00	3.01	1.00
SVRG	1.29	40.00	3.03	1.00

SAG is very sensitive to step size and starting value. The setup requires manual tuning of step size and starting values which is cumbersome.

Experiment 3: Starting value $\beta_0 = (0, 0)^\top$ when n=100 and n=1000. Fix step size to 0.029 for all methods

The computation time is higher with the increase in sample size, but the number of step it takes for convergence does not change much.

Table 3: Compare $\hat{\beta}$ estimates obtained by Linear Regression with Gradient Descent, SGD as well as SAG, SAGA and SVRG when sample size, n=100

method	time	steps	beta0	beta1
LR	0.00	0.00	3.02	1.02
GD	0.79	157.00	2.99	1.02
SGD-1	1.88	697.00	3.04	1.02
SGD-10	0.68	205.00	3.04	1.02
SGD-500	0.81	157.00	2.99	1.02
SAG	1.72	692.00	3.03	1.03
SAGA	1.16	466.00	3.04	1.02
SVRG	0.93	157.00	2.99	1.02

Table 4: Compare $\hat{\beta}$ estimates obtained by Linear Regression with Gradient Descent, SGD as well as SAG, SAGA and SVRG when sample size, n=1000

	method	time	steps	beta0	beta1
LR	0.00	0.00	2.97	1.00	
GD	6.14	156.00	2.94	1.00	
SGD-1	13.90	771.00	2.97	1.00	
SGD-10	3.07	176.00	2.96	1.00	
SGD-500	3.04	163.00	2.95	1.00	
SAG	6.16	310.00	2.92	1.00	
SAGA	2.92	151.00	2.97	1.00	
SVRG	5.18	156.00	2.94	1.00	

Experiment 4: Effect of changing the starting value when n=1000 and fixed step size of 0.029 for all methods. Starting values used are $(1,1)^\top$, $(2.5,0.5)^\top$

Table 5: Compare $\hat{\beta}$ estimates obtained by SVRG when sample size, and different starting values

start	method	time	steps	beta0	beta1
$(1,1)^\top$	SVRG	4.86	142	2.94	1.00
$(2.5,0.5)^\top$	SVRG	3.21	94	2.94	1.00

Both SAG and SAGA diverges, with the setup that worked for starting value of $(0,0)^\top$. Tuneup for this starting value need to be performed again for SAG and SAGA. Closer the starting value to optimum, faster the time to convergence like expected in SVRG.

Experiment 5: Changing update frequency of SVRG when n=1000

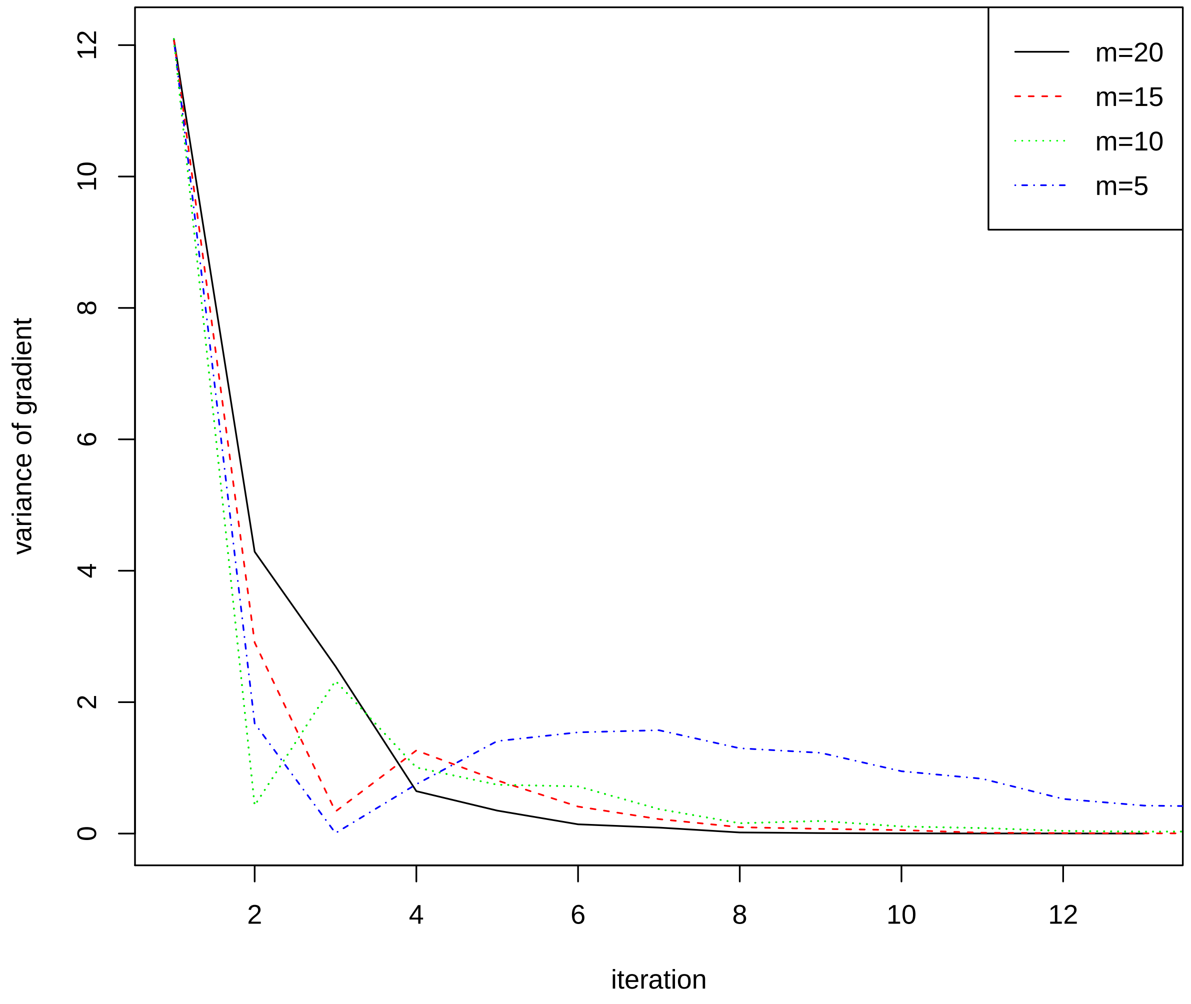
Table 6: Compare $\hat{\beta}$ estimates obtained by SVRG when sample size, and different values for the update frequency, m

m	method	time	steps	beta0	beta1
-	LR	-	-	3.0180	0.9862
3	SVRG	4.28	77	2.99	0.99
5	SVRG	2.45	47	2.99	0.99
10	SVRG	1.24	24	2.99	0.99
15	SVRG	0.90	17	2.99	0.99
20	SVRG	0.66	13	2.99	0.99

As the update frequency increases, computation time decreases and convergence happens faster.

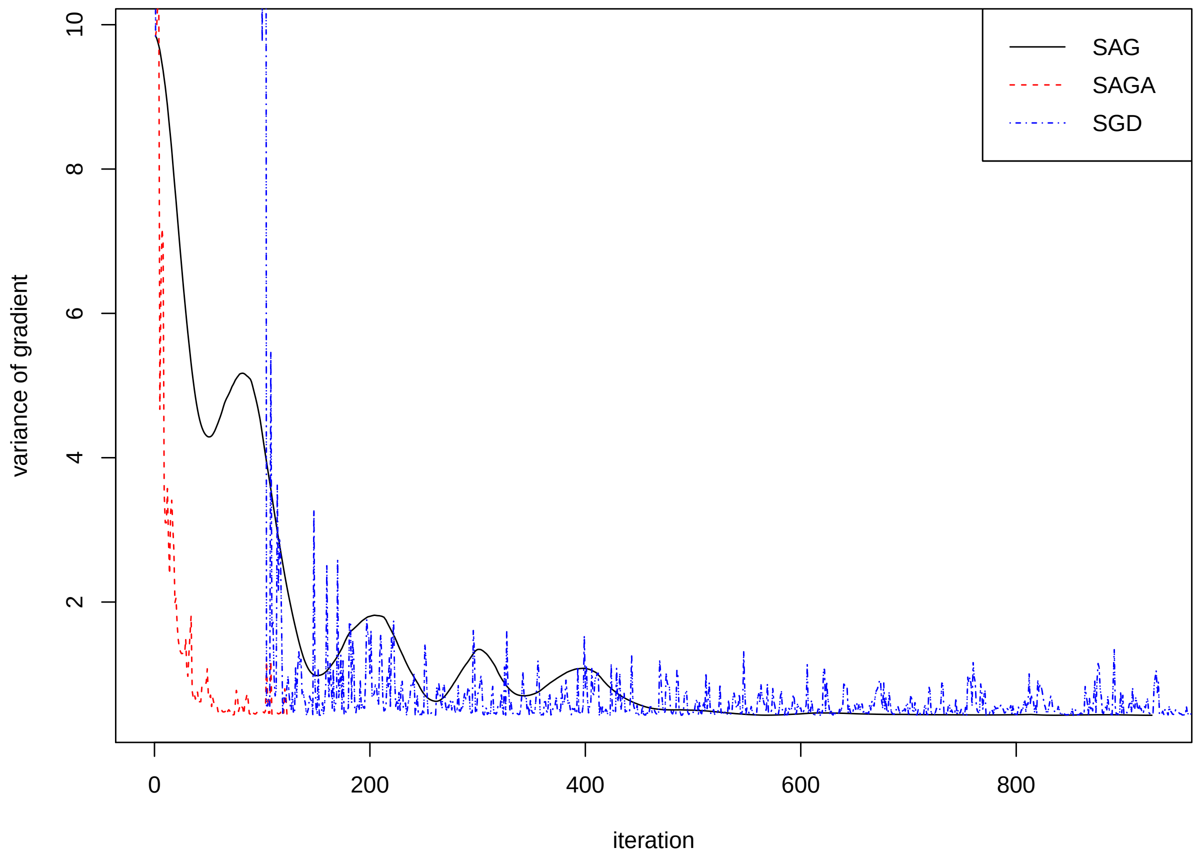
Experiment 6: Comparing how variance diminishes across different methods

Figure 2: Effect of updating frequency on variance in SVRG



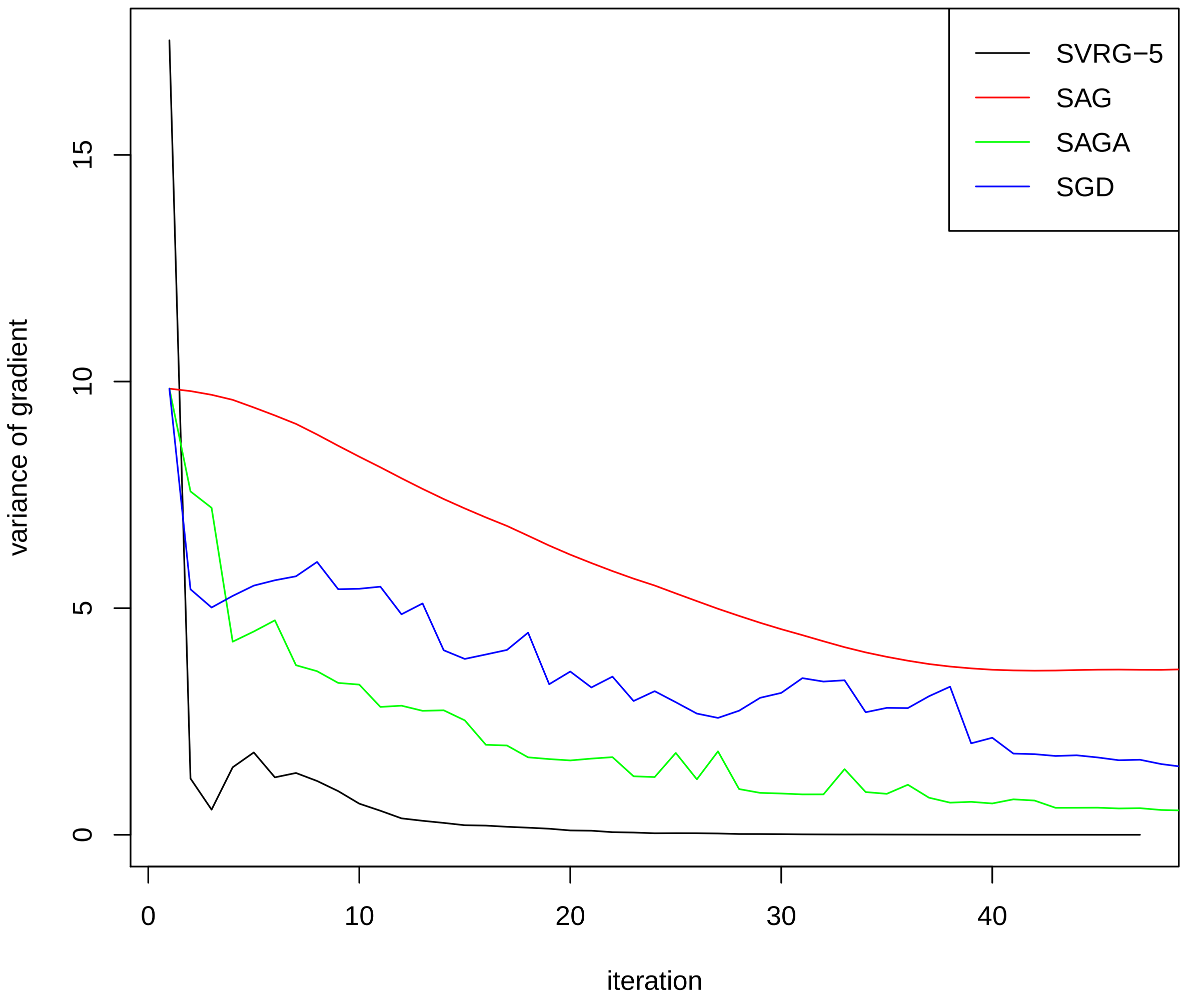
As the update frequency increases, the variance of SVRG updates are smaller and diminishes faster.

Figure 3: Diminishing variance in SAG, SAGA and SGD



The variance change for SAG is more smooth, but much higher than SAGA and SGD. SAGA has higher variance compared to SAG indicated as jagged lines. SGD has the highest variance.

Figure 4: Diminishing variance in SVRG, SAG, SAGA and SGD



The change is variance for SAG is very smooth whereas SGD is very jagged. SAGA has a more higher but lower variance due lack of smoothness. The variance of SVRG diminished faster but is not smooth like SAG. It has the least variance.

4 Conclusion

This report is a study of some of the popular variance reduction techniques for stochastic gradient descent like SAG[3], SVRG[5], and SAGA [4]. SVRG performed better compared to other methods for the problem of minimizing least squares function to estimate coefficients of linear regression problem. The update frequency has a considerable impact in reducing the variance. SAG has considerably low variance over all the techniques but is very hard to tune the step size and starting value. SAGA has a higher variance compared to SAG but lesser than SGD variance. SAGA has an advantage that it works for composite functions. Also, the proof of convergence for SVRG was simple and easy to follow unlike the proofs for SAG. SVRG has a lower memory cost compared to SAG and SAGA, since the gradients of previous updates are not stored.

There are several other methods like Stocahstic dual coordinate accent(SDCA)[7] methods exist that have not been discussed in this report. The next step would be understand the SAGA algorithm better, by designing an experiment to evaluate cases for which SAGA preforms better than SVRG, like for composite functions and nonconvex non-smooth functions.

References

[1] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[2] Coms 578x: Subgradient method. http://web.cs.iastate.edu/ jialiu/teaching/COMS578X_F18/Lectures/LN9-annotated.pdf. Accessed: 2019-12-09.

[3] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[4] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[5] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[6] Linear regression by gradient descent in r. https://www.r-bloggers.com/linear-regression-by-gradient-descent/. Accessed: 2019-12-09.

[7] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

A Code

```
#############################################################
# Two norm square
# y^T y
# @param column vector y
# @return Scalar value
##############################################################
norm <- function(y) {
  r = t(y)%*%y
  r = as.numeric(r)
  return (r)
}

#############################################################
# Gradient Descent with fixed step size of 1/sqrt(k)
# where k is the iteration number
# @param func: Function subroutine
# @param derv: derivative function subroutine
# @param start: start position
# @param s : step size
##############################################################
gradient_descent<-function(func, derv, start, s, tol=0.001, maxiters=5000){
  x1 = start
  k=1
  ###########
  ftrace <- c()
  xtrace <- list()
  ftrace <- rbind(ftrace, func(x1))
  xtrace[[k]] <- x1
  k=k+1
  ###########
  while (norm(derv(x1)) > tol) {
    df <- derv(x1) #cost of finding gradient is O(n)
    x1 <- x1 - s * df
    ##########
    ftrace <- rbind(ftrace, func(x1))
    xtrace[[k]] <- x1
    ############
    k=k+1
    if (k==maxiters) {
      break;
    }
  }

  data.frame(
    "f_x" = ftrace,
    "x" = I(xtrace),
    "k" = c(1:length(ftrace))
  )
}

#############################################################
# Stochastic Gradient Descent
# @param func: Function subroutine
# @param derv: derivative function subroutine
# @param start: start position
# @param s : step size
##############################################################
stochastic_gradient_descent<-function(func, derv, derv_i, start, n, m=1, tol=0.001, maxiters=1000){
  x1 = start
  k=1
  ###########
  ftrace <- c()
  vtrace <- c()
  xtrace <- list()
  ftrace <- rbind(ftrace, func(x1))
  vtrace<-rbind(vtrace, var(derv(x1)))
  xtrace[[k]] <- x1
  k=k+1
  ###########

  while (norm(derv(x1)) > tol) {
    s = 1/sqrt(k)
    ik = sample(1:n,m,replace=F)
    x1 <- x1 - s * derv_i(ik,x1) #cost of finding gradient is O(1)
    ##########
    ftrace <- rbind(ftrace, func(x1))
    vtrace<-rbind(vtrace, var(derv(x1)))
    xtrace[[k]] <- x1
    ############
    k=k+1
    if (k==maxiters) {
      break;
    }
  }

  data.frame(
    "f_x" = ftrace,
    "v_x" = ftrace,
    "x" = I(xtrace),
    "k" = c(1:length(ftrace))
  )
}

#############################################################
# Stochastic Average Gradient: SAG
# @param func: Function subroutine
# @param derv: derivative function subroutine
# @param start: start position
# @param s : step size
##############################################################
sgd_sag<-function(func, derv, derv_i, start, n, m=1, s = 0.01, tol=0.01, maxiters=2000){
  x1 = start
  k=1
  g = list()
  for  (i in 1:n) {
    g[[i]] <- x1
  }
  ###########
  ftrace <- c()
  vtrace <- c()
  xtrace <- list()
  ftrace <- rbind(ftrace, func(x1))
  vtrace<-rbind(vtrace, var(derv(x1)))
  xtrace[[k]] <- x1
  k=k+1
  ###########

  while (norm(derv(x1)) > tol) {
    ik = sample(1:n,m,replace=T)
    sumg0<- matrix(rep(0, length.out=length(x1)), nrow=length(x1),ncol=1)
    for  (i in 1:n) {
      sumg0 = sumg0+g[[i]]
    }
    g0 = g[[ik]]
    g1 = derv_i(ik, x1)
    g[[ik]]=g1

    x1 <- x1 - s * (g1 - g0 + sumg0)

    ##########
    ftrace <- rbind(ftrace, func(x1))
    vtrace<-rbind(vtrace, var(derv(x1)))
    xtrace[[k]] <- x1
    ############
    k=k+1
    if (k==maxiters) {
      break;
    }
  }
  data.frame(
    "f_x" = ftrace,
    "v_x" = ftrace,
    "x" = I(xtrace),
    "k" = c(1:length(ftrace))
  )
}

#############################################################
# Stochastic Gradient Descent: SAGA
# @param func: Function subroutine
# @param derv: derivative function subroutine
# @param start: start position
# @param s : step size
##############################################################
sgd_saga<-function(func, derv, derv_i, start, n, m=1, s=0.01, tol=0.001, maxiters=1000){
  x1 = start
  k=1
  g = list()
  for  (i in 1:n) {
    g[[i]] <- x1
  }

  ###########
  ftrace <- c()
  vtrace <- c()
  xtrace <- list()
  ftrace <- rbind(ftrace, func(x1))
  vtrace<-rbind(vtrace, var(derv(x1)))
  xtrace[[k]] <- x1
  k=k+1
  ###########

  while (norm(derv(x1)) > tol) {
    ik = sample(1:n,m,replace=F)
    sumg0<- matrix(rep(0, length.out=length(x1)), nrow=length(x1),ncol=1)
    for  (i in 1:n) {
      sumg0 = sumg0+g[[i]]
    }
    g0 = g[[ik]]
    g1 = derv_i(ik, x1)
    g[[ik]]=g1
    x1 <- x1 - s * (g1*n - g0*n + sumg0)
    ##########
    ftrace <- rbind(ftrace, func(x1))
    vtrace<-rbind(vtrace, var(derv(x1)))
    xtrace[[k]] <- x1
    ############
    k=k+1
    if (k==maxiters) {
      break;
    }
  }

  data.frame(
    "f_x" = ftrace,
    "v_x" = ftrace,
    "x" = I(xtrace),
    "k" = c(1:length(ftrace))
  )
}

#############################################################
# Stochastic Gradient Descent: SVRG
# @param func: Function subroutine
# @param derv: derivative function subroutine
# @param start: start position
# @param s : step size
##############################################################
sgd_svrg<-function(func, derv, derv_i, start, n, m=1, s=0.01, tol=0.001, maxiters=1000){
  x1 = start
  k=1
  ###########
  ftrace <- c()
  vtrace <- c()
  xtrace <- list()
  ftrace <- rbind(ftrace, func(x1))
  vtrace<-rbind(vtrace, var(derv(x1)))
  xtrace[[k]] <- x1
  ###########
  k=k+1

  while (norm(derv(x1)) > tol) {
    g0 = derv(x1)
    ibatch = sample(1:n, m, replace = F)
    x0 = x1
    for (i in ibatch) {
      x1 <- x1 - s*(n*derv_i(i, x1) - n*derv_i(i, x0) + g0)
    }
    ##########
    ftrace <- rbind(ftrace, func(x1))
    vtrace<-rbind(vtrace, var(derv(x1)))
    xtrace[[k]] <- x1
    ############
    k=k+1
    if (k==maxiters) {
      break;
    }
  }

  data.frame(
    "f_x" = ftrace,
    "v_x" = vtrace,
    "x" = I(xtrace),
    "k" = c(1:length(ftrace))
  )
}

##############################################################################
#########################################
# Function: squared error cost function
#########################################
cost <- function(theta) {
  c <- 0
  for (i in 1:length(y)) {
    c<- c+cost_i(i,theta)
  }
  c
}

cost_i <- function(i, theta) {
  ( (X[i,] %*% theta - y[i])^2 ) / (2*length(y))
}

#######################################################
# Gradient: Derivative of squared error cost function
#######################################################
delta <- function(theta, B=c()) {
  d <- matrix(rep(0,length.out=length(theta)),nrow=length(theta),ncol=1)
  if (length(B)==0) {
    B = 1:length(y)
  }
  for (i in B) {
    d <- d + delta_i(i,theta, B)
  }
  d
}

delta_i <- function(i, theta, B=c()) {
  if (length(B)==0) {
    N = length(y)
  }
  else {
    N = length(B)
  }
  error<-as.numeric(X[i,] %*% theta - y[i])
  t(error*t(X[i,])/N)
}
########################################################################

#Simulate data
x <- runif(100, -5, 5) #n=1000 or 100
y <- x + rnorm(100) + 3

# fit a linear model
lrfit <- lm( y ~ x )

# plot the data and the model
plot(x,y, col="grey")
abline(res, col='blue')

# starting value
beta0 <- matrix(c(0,0), nrow=2)

# design matrix
X <- cbind(1, matrix(x))

##########################################
#Plot variance updates
y4 = sgd_svrg(cost,delta, delta_i, start = beta0, n=length(y), m=5, s=0.02, tol=0.001)
y3 = sgd_sag(cost,delta, delta_i, beta0, length(y), m=1, s=0.029, tol=0.001)
y2 = sgd_saga(cost,delta, delta_i, beta0, length(y), m=1, s=0.03, tol=0.001)
y1 = stochastic_gradient_descent(cost,delta, delta_i, beta0, length(y), m=1)
plot(y4$v_x, type="l", lty=1, col="black", xlab = "iteration", ylab="variance of gradient")
lines(y3$v_x, lty=2, col="red")
lines(y2$v_x, lty=3, col="green")
lines(y1$v_x, lty=4, col="blue")
legend("topright", c("SVRG-5","SAG","SAGA","SGD"), col=c("black","red", "green","blue"), lty=c(1,2,3,4)
############################################
```