Semi Stochastic Methods - Mode: Application

Pragyan Pandey
Electrical Engineering
Indian Institute of Technology Kanpur
17807478
Email: pragyan@iitk.ac.in

***Abstract*—A number of problems in the field of Data Science, Machine Learning, Biology, Economics and various other fields can be expressed in the form of minimizing some convex loss function. In this paper, we take a look problem of minimizing convex loss functions using Semi-Stochastic Gradient Descent(S2GD), Mini-Batch Semi-Stochastic Gradient Descent(mS2GD) and Semi-Stochastic Coordinate Descent(S2CD).**

I. Introduction

We look at three convex optimization algorithms - Semi-Stochastic Gradient Descent, Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting and Semi-Stochastic Coordinate Descent. In all three methods, first we formulate the problem or type of convex problems taken into consideration for each method in detail. We individually analyze the three methods with respect to older methods like Stochastic Gradient Descent and finally we discuss the performances of the algorithm with each other.

II. Semi-Stochastic Gradient Descent Methods

A. Introduction

The author studies the problem of minimizing average of n(where n is very large) smooth convex loss functions. The author proposes Semi-Stochastic Gradient Descent(S2GD) which runs for a single or several epochs and in every epoch we compute a single full gradient and a random number of stochastic gradients using a geometric law.

B. Problem Formulation and Motivation

The optimization problem can be cast of the form:

$$\min_{\mathbf{x}\in R^D} f(\mathbf{x}) \tag{1}$$

where

$$f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{2}$$

N denotes the number of training examples, D denotes the dimension of $\mathbf{x}$ or number of features of feature vector and $f_i(\mathbf{x})$ denotes the loss for the i^{th} training example.$f_i(\mathbf{x})$ are strongly convex loss functions. Our aim is to find the predictor $\mathbf{x} \in R^D$ such that it minimizes the average loss $f(\mathbf{x})$. Typically N is very large i.e. N>>D. Also it should be noted that $f(\mathbf{x})$ includes all the regularization functions as well.

The optimization problem proposed above can be solved by using Gradient Descent. For k^{th} iteration:

$$\mathbf{x}_k = \mathbf{x_{k-1}} - hf'(\mathbf{x}_{k-1}) \tag{3}$$

where h is the stepsize and $f'(\mathbf{x}_{k-1})$ is the full gradient of f at $\mathbf{x}_{k-1}$. Computing $f'(\mathbf{x}_{k-1})$ involves computing gradients of N f_i's and since we know N is big, this can be rather cumbersome task, making the algorithm too slow for all practical purposes. To speed up the process, we use Stochastic Gradient Descent(SGD), which instead of calculating the full gradient $f'(\mathbf{x}_{k-1})$ picks a random number i and performs the updates of the form:

$$\mathbf{x}_k = \mathbf{x_{k-1}} - hf_i'(\mathbf{x}_{k-1}) \tag{4}$$

We can observe that this reduces the time complexity by a factor of N. $f_i'(\mathbf{x}_{k-1})$ are referred to as Stochastic Gradients. While the algorithm is faster but a practical issue with SGD is that consecutive stochastic gradients may vary a lot so much so they may point in opposite direction. This has a major impact on the performance of the algorithm.

To handle both the problems of speed in GD and accuracy in SGD, the author proposes Semi-Stochastic Gradient Descent which has the benefits of both the algorithms:the speed of SGD and the accuracy of GD.

C. Algorithm

The author proposes S2GD and S2GD+ and analyzes the former only. Also assumes that all f_i are convex and L smooth.

1) S2GD: S2GD mainly depend on three parameters - stepsize h, m is the limiting number of stochastic gradients computed in a single epoch and $\nu \in [0, \mu]$, where μ is the strong convexity constant of f.

Algorithm 1:Semi-Stochastic Gradient Descent(S2GD)

parameters:h - stepsize, m-max # stochastic gradients for single epoch, ν lower bound on μ.
for j = 0,1,2,...**do**
$g_j \leftarrow \frac{1}{N}\sum_{i=1}^{N} f_i'(\mathbf{x}_j)$
$y_{j,0} \leftarrow \mathbf{x_j}$

Let $t_j \leftarrow t$ with probability $\frac{(1-\nu h)^{m-t}}{\beta}$ for t = 1,2....,m
for t = 0 to t_j-1 **do**
Pick $i \in \{1, 2....N\}$, uniformly at random
$y_{j,t+1} \leftarrow y_{j,t} - h(g_j + f_i'(y_{j,t}) - f_i'(\mathbf{x}_j))$
end for
$\mathbf{x}_j \leftarrow y_{j,t_j}$
end for

The algorithm contains an outer loop indexed by j and an inner loop on t. For every epoch j, a full gradient g_j is computed and then produces a random number $t_j \in [1, 2..., m]$ following a geometric law, where

$$\beta = \sum_{i=1}^{m} (1 - \nu h)^{m-t} \tag{5}$$

and only two stochastic gradients are computed in each step. For each j, the expected number of iterations($E\{t_j\}$) are:

$$E\{t_j\} = \sum_{t=1}^{m} t \frac{(1 - \nu h)^{m-t}}{\beta} \tag{6}$$

Note: $E\{t_j\} \in [\frac{m+1}{2}, m)$. The lower bound $\frac{m+1}{2}$ is achieved when ν = 0 and the upper bound when $h\nu \to 1$.

Algorithm 2: S2GD+

parameters: $\alpha \geq 1$
Run SGD for a single pass over the data; (returns $\mathbf{x}$)
Starting from $\mathbf{x}_0 = \mathbf{x}$, run a version of S2GD in which $t_j = \alpha n$ for all j

The performance of S2GD+ is better than S2GD. It starts by running SGD and makes a single pass over the whole data. As we know, SGD computes only one stochastic gradient for every update of $\mathbf{x}$ as compared to GD in which full gradients have to computed. The first step of S2GD involves computing full gradient of f. Hence, starting from SGD and then moving onto S2GD gives us a superior method.

D. Time Complexity

1) Complexity for strongly convex f: If f is strongly convex S2GD takes $O((N + \kappa) log(\frac{1}{\epsilon}))$ where N is the total number of training examples, $\kappa = \frac{L}{\mu}$ is the condition number and we want ϵ-approximate solution. S2GD achieves this complexity for stepsize $h = O(1/L)$, $j = O(log(1/\epsilon))$ epochs and $m = O(\kappa)$.

2) Complexity for convex f: If f is not strongly convex then with strong convexity constant $\mu = O(\frac{L}{\epsilon})$ is

$$O((N + \frac{L}{\epsilon}) log(1/\epsilon)) \tag{7}$$

which is better than standard rate of SGD.

E. Numerical Experiments

The author conducts various experiments on the actual datasets and analyzes the performance of the algorithm S2GD and compares them with other well known algorithms.

Dataset	Training Exam-ples(N)	D	L	μ	κ
ijcnn	40990	23	1.23	1/N	61696
rcvl	20242	47237	0.50	1/N	10122
real-sim	72309	20959	0.50	1/N	36155
url	2396130	3231962	128.70	100/N	3084052

TABLE I
DATASETS USED IN THE EXPERIMENT

1) Comparison with theory: In this experiment author takes a artificial dataset, with a condition number that we can control. The function f_i's are of the form:

$$f_i(\mathbf{x}) = \frac{1}{2}(\mathbf{a}_i^T \mathbf{x} - b_i) + \frac{\lambda}{2}||\mathbf{x}||_2^2 \tag{8}$$

which is our L2-regularized least squares with $\mathbf{a}_i \in R^D, b_i \in R$ and $\lambda > 0$.

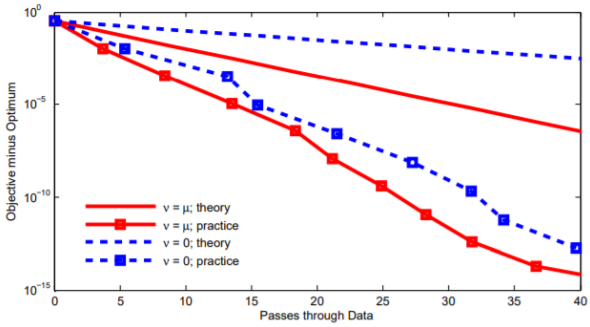

Fig. 1. Least squares with N = 10^5 , κ = 10^4 . Comparison of theoretical result and practical performance for cases $\nu = \mu$ (full red line) and $\nu = 0$ (dashed blue line).

We take N = 10^5, $\kappa = 10^4$, $D = 1000$. We run the algorithm for both $\nu = \mu$ and $\nu = 0$. The practical performance demonstrated is over a single run. Figure 1 demonstrates that S2GD shows linear convergence in practice even better than the theoretical results. This method is also an improvement over [4].

2) Comparison with other methods: In this experiment, the author takes L2-regularized loss function used for binary classification. Also multiple datasets are considered for the experiment so that we have observations that are more general. The functions f_i's are of the form:

$$f_i(\mathbf{x}) = log(1 + exp(l_i \mathbf{a}_i^T \mathbf{x})) + \frac{\lambda}{2}||\mathbf{x}||_2^2 \tag{9}$$

where l_i is the label of the i^{th} training example $\mathbf{a}_i$. λ is set to be $O(\frac{1}{N})$ and the condition number $\kappa = O(N)$.

All the datasets used are listed in Table [1] and are freely available.

In the experiment, the author compares the following algorithms - **SGD** (Stochastic Gradient Descent), **L-BFGS** - a publicly available limited- memory quasi-Newton method,

SAG(Stochastic Average Gradient) , **S2GDcon**- this is S2GD with stepsizes that are suggested by the theory and finally **S2GD** with stepsize that gives the best performance.

We plot the performance of all the algorithm for 15-20 passes over the data in figure [2] and figure [3].

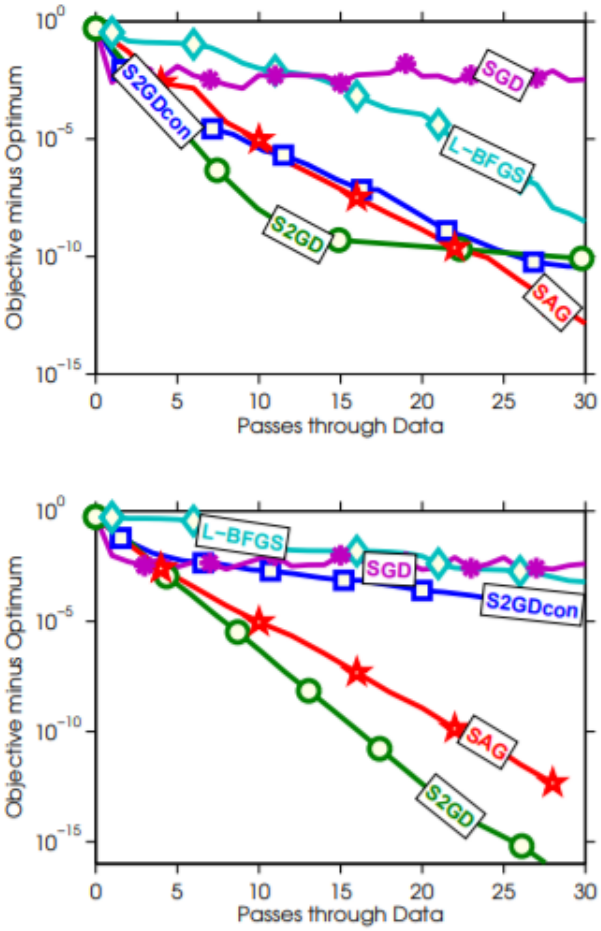

Fig. 2. Practical performance for logistic regression and the following datasets: ijcnn, realsim(second)

Observations:
There is a performance gap between S2GD and S2GDcon that is typically dataset dependent. It is also evident that S2GD in general performs better than all other algorithms. The author compares the performance of the algorithms using the number of passes through the data. As we can see that some algorithms require to more time to make the same number of passes through data than others. We can also observe that S2GD is in fact faster than SAD as SAG updates the test point after computing each stochastic gradient while S2GD does not always update the test point.

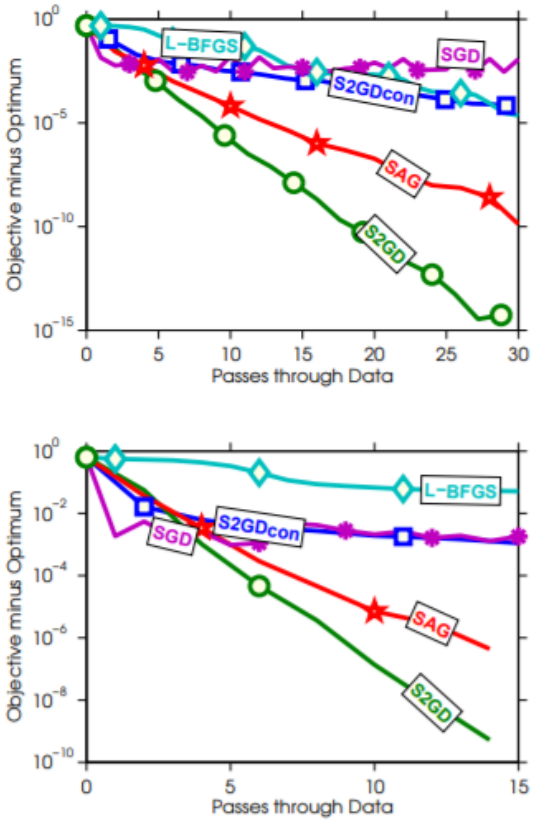

Fig. 3. Practical performance for logistic regression and the following datasets: rcv, url(second)

III. Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting

A. Introduction

In this paper author again considers the same problem of minimizing convex functions with minor modification as compared to the above paper.

$$min_{\mathbf{x}\in R^D}\{P(\mathbf{x}) := F(\mathbf{x}) + R(\mathbf{x})\}, \tag{10}$$

where F is a smooth function and R is non-smooth (used to account for constraints in optimization). F can be further decomposed into an average of a large number of loss functions i.e.

$$F(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{11}$$

B. Motivation

The author argues that while the modern stochastic methods like S2GD do not have the "non-vanishing variance" problem (we explained this while discussing SGD in [1]) as compared to SGD, hence it does need any mini batch treatments but still mini batch are still quite useful. The author proposes mS2GD which is the mini-batch treatment of S2GD. While S2GD algorithm that we saw in [1] is a special case when **R**=0.

The Mini-batch S2GD has various benefits when compared to other modern stochastic methods.The algorithm runs faster in clock time in HPC(High-performance computing) environment which is critical to its performance when datasets are huge and is one of the main advantages and motivation. Further, the mS2GD attains ϵ-approximate solutions faster than S2GD.

C. Algorithm

The author first motivates the mathematical setup of deterministic and stochastic gradient methods for the algorithm.

1) Deterministic and stochastic proximal gradient methods: The classical approach to solve our optimization problem is forming a sequence $\{y_t\}$ using

$$y_{t+1} = argmin_{x \in R^D} U_t(\mathbf{x}) \tag{12}$$

where $U_t(\mathbf{x})$ is defined as:

$$U_t(\mathbf{x}) = F(\mathbf{y}_t) + \nabla F(\mathbf{y}_t)^T(\mathbf{x} - \mathbf{y}_t) + \frac{1}{2h}||\mathbf{x} - \mathbf{y}_t||_2^2 + R(\mathbf{x}) \tag{13}$$

where $U_t(\mathbf{x})$ is an upper bound P whenever stepsize parameter satisfies $h \leq \frac{1}{L}$. The above expression can be compactly written using the proximal operator as:

$$y_{t+1} = prox_{hR}(\mathbf{y}_t - h\nabla G_t) \tag{14}$$

where G_t is the stochastic estimate of the gradient $\nabla F(\mathbf{y}_t)$ and proximal operator is defined as

$$prox_{hR}(\mathbf{z}) = argmin_{\mathbf{x} \in R^D}\{\frac{1}{2}||\mathbf{x} - \mathbf{z}||_2^2 + hR(\mathbf{x})\} \tag{15}$$

2) Mini-Batch Semi Stochastic Gradient Descent: Now the author describes the algorithm:-

---

Algorithm: mS2GD

---

parameters: stepsize h, m - max # of stochastic steps per epoch, $\mathbf{x}_0$ starting point, $b \in R$ - mini batch size
for k = 0,1,2....N **do**
Compute and store $g_k \leftarrow \nabla F(\mathbf{x}_k) = \sum_{i=1}^N \frac{1}{N} f_i'(\mathbf{x}_k)$
$\mathbf{y}_{k,0} = \mathbf{x}_k$
Choose $t_k \in \{1, 2, ..., m\}$ uniformly at random
for t = 0 to $t_k - 1$ **do**
Choose mini-batch $A_{kt} \subset [N]$ of size b, uniformly at random
Compute stochastic estimate of $\nabla F(\mathbf{y}_{k,t})$
$G_{k,t} \leftarrow g_k + \frac{1}{b}\sum_{i \in A_{k,t}}(\nabla f_i(\mathbf{y}_{k,t}) - \nabla f_i(\mathbf{x}_k))$
$\mathbf{y}_{k,t+1} \leftarrow prox_{h,R}(y_{k,t} - hG_{k,t})$
end for
Set $\mathbf{x}_{k+1} \leftarrow \mathbf{y}_{k,t_k}$
end for

In the algorithm, we have a outer loop which is indexed by k (epoch counter) and an inner loop which is indexed by t_k which is uniformly chosen from {1,2,3...m}. In each epoch, first we compute the full gradient of F at $\mathbf{x}_k$ and then proceed to the inner loop and perform proximal update for every inner loop iteration, however with stochastic estimate of gradient $G_{k,t}$, which we do using mini-batch $A_{k,t} \in [N]$, makes each iteration run for time 2b where b is the size of mini-batch $A_{k,t}$.

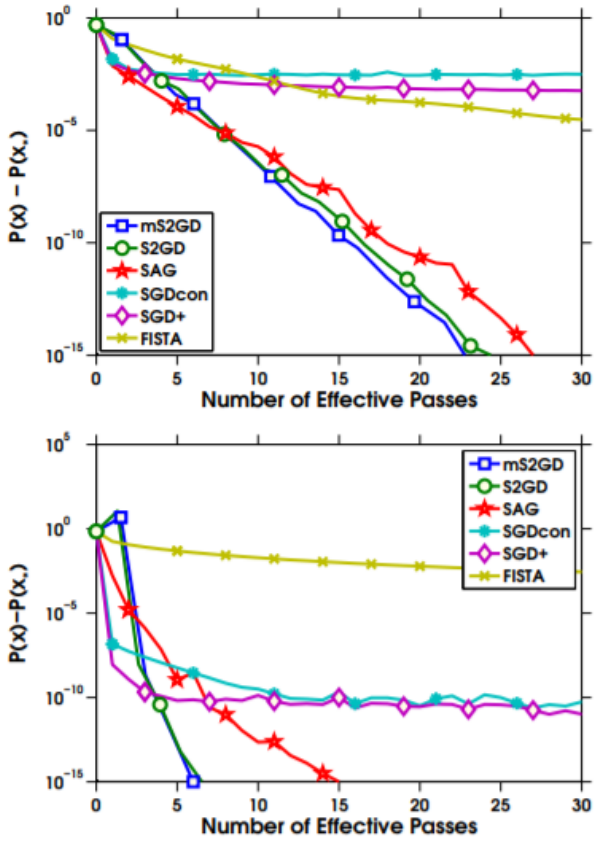

Fig. 4. Comparison of several algorithms: rcvl(top), convtype(bottom)

D. Numerical Results

In this section, author performs the numerical experiments to illustrate the performance of the algorithm. **mS2GD**(with batch size b = 8) is compared with the **S2GDcon**, **S2GD**, **SGD+** - where stepsize h = $\frac{h_0}{k+1}$ is variable as compared SGD, **SAG** and **FISTA** - Fast iterative shrinkage-thresholding algorithm. Experiment is conducted with $R(\mathbf{x}) = \frac{\lambda}{2}||\mathbf{x}||_2^2$ and F is of the form:

$$F = \frac{1}{N}\sum_{i=1}^N f_i(\mathbf{x}) \tag{16}$$

where

$$f_i(\mathbf{x}) = log[1 + exp(-b_i \mathbf{a}_i^T \mathbf{x})] \tag{17}$$

The b_i are the labels from the set {-1,1} and $\mathbf{a}_i \in R^D$. Hence our optimization function $P(\mathbf{x})$ is:

$$P(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^N log[1 + exp(-b_i \mathbf{a}_i^T \mathbf{x})] + \frac{\lambda}{2}||\mathbf{x}||_2^2 \tag{18}$$

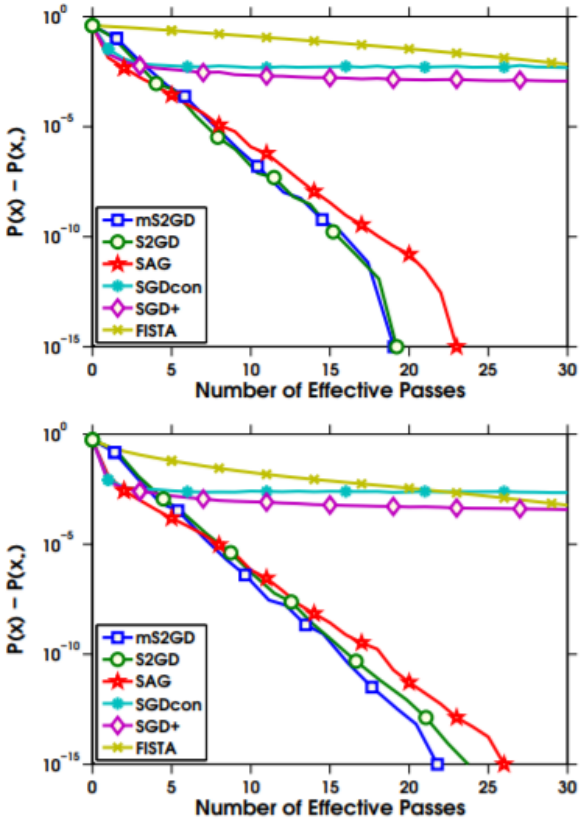

Fig. 5. Comparison of several algorithms:news20(top), astrophy(bottom)

Dataset	Training Exam-ples(N)	D	Sparsity	L
news20	19996	1355191	0.0336%	0.2500
rcvl	20242	47237	0.1569%	0.2500
covtype	581012	54	22.1212%	1.9040
astro-phy	62369	99757	0.0767%	0.2500

TABLE II
DATASETS USED IN THE EXPERIMENT

The regularization parameter λ is set to $\frac{1}{N}$ making the condition number $\kappa = \frac{L}{\mu}$ to be of $O(N)$.

In Table 2, all the datasets are summarized including number of training examples N, Dimension of $\mathbf{x}$ D and Lipschitz constants L.

Observations:
It is evident that performance of mS2GD is significantly better than other algorithms. The algorithm performs slightly better than S2GD but one more significant advantage in mS2GD is that if parallel processors are available, we can significantly increase even this performance which is not the case with other algorithms.

IV. Semi-Stochastic Coordinate Descent

A. Introduction

In this paper, the author proposes a novel gradient descent algorithm - Semi Stochastic Coordinate Descent (S2CD) for the problem of minimizing average of a large number of smooth convex functions: $f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$. The novelty of this algorithm lies in computing of stochastic gradient step where a random function f_i and a random coordinate j - using non uniform distributions and update only a single index of the decision vector, based on j^th partial derivative of f_i at different points.

B. Problem Formulation and Assumptions

The paper studies the problem of unconstrained minimization of an average of a large number of strongly convex functions.

$$min_{\mathbf{x}\in R^D} f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{19}$$

In machine learning, $f_i(\mathbf{x})$ is loss for i^{th} training example. f denotes the average loss. Sometimes $\mu||\mathbf{x}||_2^2$ is also added to f making strongly which also makes this easier to minimize.

Assumptions:We assume that $f_i : R^D \to R$ are differentiable and convex functions, with Lipschitz continuous partial derivatives. Formally, we assume that for each $i \in [N]$ and $j \in [D]$, there exists $L_{ij} \geq 0$ such that for $\mathbf{x} \in R^D$ and $h \in R$

$$f_i(\mathbf{x} + h\mathbf{e_j}) \leq f_i(\mathbf{x}) + \langle \nabla f_i(\mathbf{x}), h\mathbf{e_j}\rangle + \frac{L_{ij}}{2}h^2 \tag{20}$$

where $\langle .,. \rangle$ is standard inner product, $\mathbf{e_j}$ is the j^{th} standard basis vector in R^D and ∇f_i is the gradient of f_i at point $\mathbf{x}$. Author further assumes that all f_i's are strongly convex.

C. Algorithm

We describe the algorithm S2CD in this section.

Algorithm: Semi Stochastic Coordinate Descent(S2CD)

parameters: $h > 0$ stepsize, m - of stochastic steps in each epoch, $\mathbf{x}_0 \in R^D$-the starting point
for k = 1,2,.. **do**
Compute and store $\nabla f(\mathbf{x}_k) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(\mathbf{x})$
$\mathbf{y}_{k,0} \leftarrow \mathbf{x}_k$
Choose t_k from {1,2,...,m} with probability $\frac{(1-\mu h)^{m-t}}{\beta}$
for t = 0 to $t_k - 1$ **do**
Choose a coordinate j from 1,2,...,D with probability p_j
Choose a i from the set $\{i{:}L_{ij} > 0\}$ with probability q_i
Update the j^{th} coordinate:
$\mathbf{y}_{k,t+1} \leftarrow \mathbf{y}_{k,t} - hp_j^{-1}(\nabla_j f_i(\mathbf{x}_k) + \frac{1}{q_{ij}}(\nabla_j f_i(\mathbf{y}_{k,t}) - \nabla_j f_i(\mathbf{x}_k)))\mathbf{e}_j$
end for
$\mathbf{x}_{k+1} \leftarrow \mathbf{y}_{k,t_k}$
end for

The algorithm has an outer loop indexed by k and an inner loop that is indexed by t. At the start of every outer loop, gradient of f is calculated at $\mathbf{x}_k$. The number t_k is chosen randomly using a geometric law:

$$P(t_k = T) = \frac{(1-\mu h)^{m-T}}{\beta} \tag{21}$$

where

$$\beta = \sum_{t=1}^{m}(1-h\mu)^{m-t} \tag{22}$$

We calculate $\mathbf{y}_{k,t+1}$ given $\mathbf{y}_{k,t}$ in each inner loop iteration. We draw a sample coordinate j from probability distribution p_j and a sample index i from probability distribution q_{ij}, the probability distributions are as follows:

$$w_i := |\{j : L_{ij} \neq 0\}|, \qquad v_j = \sum_{i=1}^{N} w_i L_{ij},$$

$$p_j := \frac{v_j}{\sum_{j=1}^{D} v_j}, \qquad q_{ij} = \frac{w_i L_{ij}}{v_j}, \qquad p_{ij} = p_j q_{ij}$$

Here w_i denotes the coordinates on which function f_i depends upon. Note that p_{ij} is the probability of choosing pair (i, j). S2CD can be seen as a hybrid between S2GD and CD. The complexity of the algorithm is sum of terms:

$$O(Nlog(\frac{1}{\epsilon}))$$

evaluations ∇f_i ($log(\frac{1}{\epsilon})$) evaluations of gradient of f_i) and

$$O(\hat{k}log(\frac{1}{\epsilon}))$$

computations of $\nabla_j f_i$ for randomly chosen i and j where $\hat{\kappa}$ is the cost of evaluation of a partial derivative $\nabla_j f_i$ and is in general greater than κ. The total time complexity of S2CD to obtain a ϵ-approximate solution is

$$O((N + \hat{\kappa})log(\frac{1}{\epsilon})) \tag{23}$$

D. Numerical Experiments

In this section, we perform the numerical experiments to compare the performance of S2CD with mS2GD(batch size =10) and S2GD. We conduct the experiment with F of the form

$$F = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{24}$$

where

$$f_i(\mathbf{x}) = log[1 + exp(-b_i \mathbf{a}_i^T \mathbf{x})] \tag{25}$$

The b_i are the labels from the set {-1,1} and $\mathbf{a}_i \in R^D$. In our experiment, we have generated synthetic for N = 1000 Data Points. Other parameters - $\nu = 0.1$ and stepsize $h = 0.0001 * batchsize$. We run the program for 500 epochs for each of the algorithms.

We will also like to point out that $\hat{\kappa}$ (i.e. time needed to evaluate $\nabla_j f_i$) in our implementation of the algorithm is quite large and that is why the code for S2CD takes more time to run as compared to S2GD or mS2GD. The reason being sampling from distributions p_i and q_j repetitively.

The results of the experiment can be observed in Figure [6].

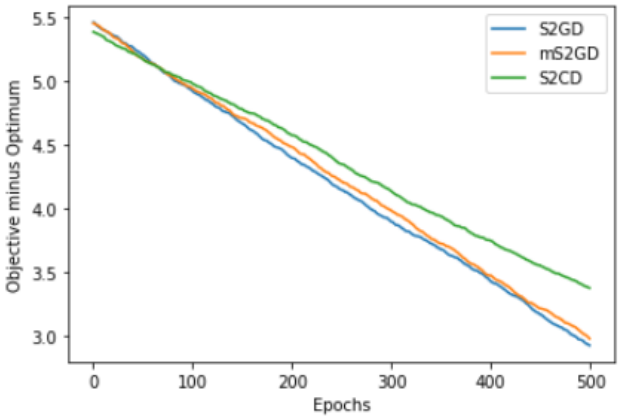

Fig. 6. Comparison of algorithms on synthetically generated data set

V. Conclusion

: We can see that the performances of the three algorithms - S2CD, S2GD and mS2GD are very close to each other. Indeed this must be the case as S2CD is a hybrid of S2GD and CD. On observing the plots closely, we can see that mS2GD performs a little better than S2GD, reason being mini-batching certainly increases the performance of S2GD and if we have multiple cores, then parallel processing could greatly increase the performance of mS2GD as compared to S2GD and S2CD. While if the data is sparse is more sparse we would observe that S2CD will perform relatively better than other two due to its property of updating single coordinate at a time. Eventually, it boils down to information about the data and how we want to model it and use these methods.

References

[1] Semi-Stochastic Gradient Descent Methods Jakub Kone˘cn´y Peter Richt´arik

[2] Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting Jakub Konecn, Jie Liu, Peter Richt, Martin Tak

[3] Semi-Stochastic Coordinate Descent Jakub Kone˘cn´y Zheng Qu Peter Richt´arik

[4] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. arXiv:1309.2388, 2013

[5] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," SIAM J. Imaging Sciences, vol. 2, no. 1, pp. 183–202, 2009.

[6] Olivier Fercoq and Peter Richt´arik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. arXiv:1309.5885, 2013.