JMLR: Workshop and Conference Proceedings 80:1–16, 2017 ACML 2017

Mini-batch Block-coordinate based Stochastic Average Adjusted Gradient Methods to Solve Big Data Problems

Vinod Kumar Chauhan VKUMAR@PU.AC.IN
Department of Computer Science & Applications,
Panjab University Chandigarh

Kalpana Dahiya KALPANAS@PU.AC.IN
University Institute of Engineering & Technology,
Panjab University Chandigarh

Anuj Sharma ANUJS@PU.AC.IN
Department of Computer Science & Applications,
Panjab University Chandigarh
Homepage: https://sites.google.com/site/anujsharma25

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

Big Data problems in Machine Learning have large number of data points or large number of features, or both, which make training of models difficult because of high computational complexities of single iteration of learning algorithms. To solve such learning problems, Stochastic Approximation offers an optimization approach to make complexity of each iteration independent of number of data points by taking only one data point or mini-batch of data points during each iteration and thereby helping to solve problems with large number of data points. Similarly, Coordinate Descent offers another optimization approach to make iteration complexity independent of the number of features/coordinates/variables by taking only one feature or block of features, instead of all, during an iteration and thereby helping to solve problems with large number of features. In this paper, an optimization framework, namely, Batch Block Optimization Framework has been developed to solve big data problems using the best of Stochastic Approximation as well as the best of Coordinate Descent approaches, independent of any solver. This framework is used to solve strongly convex and smooth empirical risk minimization problem with gradient descent (as a solver) and two novel Stochastic Average Adjusted Gradient methods have been proposed to reduce variance in mini-batch and block-coordinate setting of the developed framework. Theoretical analysis prove linear convergence of the proposed methods and empirical results with bench marked datasets prove the superiority of proposed methods against existing methods.

Keywords: Stochastic approximation, coordinate descent, stochastic gradient descent, gradient descent, block coordinate update, Big data optimization.

1. Introduction

Big data problems have multiple aspects and one of them is the size/volume of data, i.e., big data problems have large number of data points or large number of features for each data point, or both, which pose a major challenge to machine learning to train models on the large

datasets because of high computational cost of each iteration of the learning algorithms. This high iteration complexity is because of solving for all the variables (features) over all data points during each iteration of the learning algorithms. Since every computer has limited capability, each iteration might be very expensive or even infeasible to process. One another interesting point about big data is that its meaning, i.e., what is big in 'big data' is not fixed since dataset sizes are continuously growing. So to solve big data problems, we need not only efficient learning algorithms to deal with high iteration complexity but also scalable learning algorithms to tackle growing size of datasets.

This paper solves Empirical Risk Minimization (ERM) problem which consists of average of losses over all data points. For training data $\{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\}$, where $x_i \in \mathbb{R}^p$, $y_i \in \{-1, +1\}$, $\forall\ i$, l is the number of data points, p is the number of features in each data point, and l or p, or both are assumed to be large. ERM problem is given below:

$$\min_w f(w), \quad f(w) = \frac{1}{l}\sum_{i=1}^{l} L_i(w, x_i, y_i) \tag{1}$$

where $w \in \mathbb{R}^p$ is parameter vector and L_i are loss functions for $i = 1, 2, ..., l$. We assume strong convexity and smoothness of ERM problem so L2 regularization can be used as given below:

$$f(w) = \frac{1}{l}\sum_{i=1}^{l} L_i(w, x_i, y_i) + \frac{C}{2}\|w\|^2, \tag{2}$$

where C is a regularization constant. For example, for logistic regression eq.(2) can be written as:

$$\min_w f(w), \quad f(w) = \frac{1}{l}\sum_{i=1}^{l} log(1 + exp(-y_i w^T x_i)) + \frac{C}{2}\|w\|^2, \tag{3}$$

where for the sake of simplicity, regularization can be hidden inside loss function, as $L_i(w, x_i, y_i) = log(1 + exp(-y_i w^T x_i)) + \frac{C}{2}\|w\|^2,\ \forall\ i$.

1.1. Motivation

In big data problems, high iteration complexity is due to dependency on all data points (l) and on all features (p), during each iteration of the learning algorithms. To reduce high iteration complexity, one approach is to use Stochastic approximation (SA, Robbins and Monro (1951); Kiefer and Wolfowitz (1952)) which is an iterative optimization approach to solve optimization problems which can not be solved directly but using approximations. In machine learning problems, it is computationally very expensive, and might be infeasible in some cases, to use all data points in each iteration of learning algorithms. So, SA is used which makes each iteration independent of l by considering one data point or mini-batch of data points, i.e., instead of going through all data points, only one data point or mini-batch of data points are processed in each iteration which makes a large computational difference for large values of l. Similarly, remaining data points can be used in mini-batches during other iterations. Thus, SA is computationally very efficient optimization approach to deal with large-scale problems with large number of data points because of using one data point

or mini-batch of data points instead of using all data, during each iteration of approximation method. No doubt that might affect the accuracy of solution, which can be compensated by using larger mini-batches and other techniques like SAG (Schmidt et al. (2016)) and SAGA (Defazio et al. (2014)) etc. But, SA might not be suitable for problems with large number of features (in each data point), because each iteration, in spite of being independent of l, is still dependent over p. Coordinate Descent (CD, Wright (2015)) offers another optimization approach to solve large-scale problems with large number of features. CD is a recursive approach to solve optimization problems by approximate minimization along coordinate directions or coordinate hyperplanes, i.e., it recursively solves small subproblems consisting of only one or few variables. As SA makes iteration complexity independent of l, similarly, CD makes iteration complexity independent of p by considering one feature or block of features as variables, fixing the rest features and solving the reduced subproblem thus formed. Then, new one feature or block of features are selected as variables while fixing the rest and resulting subproblem is solved. This process is repeated until all subproblems covering all features are solved, then whole process is repeated until convergence. Thus, CD is a good approach to solve large-scale problems with large number of features but again might not be sufficient to solve problems with large number of data points. So it is observed that SA makes each iteration independent of l, but still, each iteration depends on p so SA is suitable for problems with large l but might not be suitable for problems with large values of p. On the other hand, CD makes each iteration independent of p, but still, each iteration depends on l so CD is suitable for problems with large values of p but might not be suitable for problems with large values of l. Thus, an optimization framework (see, Subsection 1.2) can be developed to solve big data problems, which makes use of best of SA as well as best of CD. In this framework, each iteration is independent of both l and p by considering only one data point or mini-batch of data points with one feature or block of features, i.e., each iteration solves a reduced subproblem of one feature or block of features over one data point or mini-batch of data points. Thus, whatever is the size of the problem, the iteration complexity is very low, making this framework efficient and scalable for solving big data problems. By considering suitable mini-batch of data points and a block of coordinates during each iteration, the framework can be further improved.

Let's apply the above stated ideas of SA and CD approaches to Gradient Descent (GD) method to understand how computational complexity of a single iteration is reduced to make each iteration simple and feasible. For solving problem given in eq.(1), the parameter update rule for $(k+1)^{th}$ iteration using GD method is given by

$$w^{k+1} = w^k - \alpha f^{'}(w^k) = w^k - \frac{\alpha}{l}\sum_{i=1}^{l} L_i^{'}(w^k, x_i, y_i) \tag{4}$$

where α is the step size and $L_i^{'}$ is gradient of loss function. The computational complexity of this iteration is $O(lp)$ which is dependent on l and p both thus it would be very expensive and might be even infeasible for large values of l or p, or both. GD exhibits linear convergence rate but at the expense of high iteration complexity. Using SA approach with GD method, i.e., Stochastic Gradient Descent (SGD) method (Zhang (2004)), taking only one data point

during each iteration, the subproblem and update rule are given by

$$\begin{aligned} &\min_{w} L_{i_k}(w, x_{i_k}, y_{i_k}), \quad i_k \in \{1, 2, ..., l\}, \\ &w^{k+1} = w^k - \alpha L'_{i_k}(w^k, x_{i_k}, y_{i_k}), \quad i_k \in \{1, 2, ..., l\} \end{aligned} \tag{5}$$

where i_k is randomly selected data point. In SGD, computational complexity of iteration is $O(p)$ which is much smaller than GD method as it considers only one data point and it exhibits linear convergence rate (Schmidt (2014)) but solution is less accurate as compared to GD due to variance in the gradient values of SGD and GD which can be reduced either by using mini-batches of data points (Li et al. (2014)) in each iteration or using variance reduction methods, like SAG (Schmidt et al. (2016)), SAGA (Defazio et al. (2014)), S2GD (Konečný and Richtárik (2013)) and SVRG (Johnson and Zhang (2013)) etc.
For CD approach with GD method, i.e., for Coordinate Gradient Descent (CGD) method taking only one coordinate while keeping others fixed following subproblem and update rule are obtained:

$$\begin{aligned} &\min_{w_j} f(w_j, w_{/j}), \quad j = 1, 2, ..., p, \\ &w_j^{k+1} = w_j^k - \alpha \left[f'(w_{<j}^{k+1}, w_{\geq j}^k) \right]_j, \quad j = 1, 2, ..., p, \end{aligned} \tag{6}$$

where w_j^k means j^{th} coordinate/variable of w^k, $w_{/j}^k$ means coordinates of w^k excluding j^{th}, $\left[f'(.) \right]_j$ denotes partial derivative w.r.t. j^{th} variable, $w_{<j}^{k+1}$ denotes variables of w^{k+1} which are already computed and $w_{\geq j}^k$ denotes variables of w^k that have not yet to be advanced to iteration $(k+1)$ along with j^{th} variable being updated, i.e., eq.(6) updates coordinates in Gauss-Seidel-like manner. CGD is dependent on only one variable and still exhibits linear convergence rate (Nesterov (2012); Wright (2015)). For CGD method computational complexity of iteration is $O(l)$ which is much smaller than GD method. Now, by combining SA and CD for GD method, i.e., eqs.(5) and (6), following subproblem and update rule are obtained:

$$\begin{aligned} &\min_{w_j} L_{i_k}(w_j, w_{/j}, x_{i_k}, y_{i_k}), \quad j = 1, 2, ..., p, \ i_k \in \{1, 2, ..., l\}, \\ &w_j^{k+1} = w_j^k - \alpha \left[L'_{i_k}(w_{<j}^{k+1}, w_{\geq j}^k) \right]_j, \quad j = 1, 2, ..., p, \ i_k \in \{1, 2, ..., l\}, \end{aligned} \tag{7}$$

which solves subproblem with one variable over one data point and thus have constant computational complexity of iteration as $O(1)$. Thus, combination of SA and CD provide an efficient and scalable solution to solve big data problems of any size.

1.2. Batch Block Optimization Framework (BBOF)

SA approach is used to solve the optimization problem by considering one or some of data points during each iteration which helps to reduce the iteration complexity but affects the accuracy of solution. But by considering suitable mini-batches of data points (Li et al. (2014)) in each iteration or using variance reduction techniques, like, SAG (Schmidt et al. (2016)), SAGA (Defazio et al. (2014)), S2GD (Konečný and Richtárik (2013)) and SVRG (Johnson and Zhang (2013)) etc., solutions can be improved. On the other hand, CD is used to solve the optimization problem by considering one or some of the features/variables

to formulate a reduced subproblem, which helps to reduce the iteration complexity. By considering, a suitable block of coordinates/variables in CD, during each iteration of learning algorithm, performance can be improved. Now, combining mini-batch SA approach with block CD approach, an optimization framework is formed, which is used to solve a subproblem of block of variables over a mini-batch of data points, during each iteration of learning algorithm. This framework which can be called Batch Block Optimization Framework (BBOF), is an efficient and scalable framework and can be used to solve big data problems because of its low iteration complexity. BBOF is presented by Algorithm 1. Suppose $\{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\}$ is the training set with l data points where $x_i \in \mathbb{R}^p$, $y_i \in \{-1, +1\}$ $\forall$ i, $X = (x_1, x_2, ..., x_l)$ and $Y = (y_1, y_2, ..., y_l)$. Suppose $w \in \mathbb{R}^p$ is partitioned into $(v_1, v_2, ..., v_s)$, s blocks, $v_j \in \mathbb{R}^{p_j}$, s.t. $\sum_{j=1}^{s} p_j = p$ and X is partitioned into m mini-batches $(B_1, B_2, ..., B_m)$ of size $|B_i|$ each, s.t. $\sum_{i=1}^{m} |B_i| = l$.

Algorithm 1 Batch Block Optimization Framework (BBOF)

Inputs: m = #mini-batches, s = #blocks, n = #epochs and α = step size.
Initialize: w^0
for $k = 1, 2, ..., n$ **do**
  for $i = 1, 2, ..., m$ **do**
    Randomly select one mini-batch B_i without replacement.
    for $j = 1, 2, ..., s$ **do**
      Cyclically select one block of coordinates v_j.
      Formulate a subproblem using block v_j over mini-batch B_i as given in eq.(8).
      Solve eq.(8) and update the solution for block v_j and keep the rest blocks fixed.
    end for
  end for
end for

BBOF divides the given training set into m mini-batches, divides the features into s blocks of features and runs for n epochs, each of which goes through all data points, i.e., it runs over all batches and covers all blocks of features for each mini-batch to avoid the overhead of reading data again and again. It randomly selects one mini-batch of data points B_i without replacement and cyclically selects one block of features v_j. Other sampling schemes can be tried for sampling mini-batches and blocks of features in this framework. But cyclic sampling is used for blocks of features since it would be difficult otherwise to use sparse implementations. Moreover, as pointed out in Zhao et al. (2014), methods like, SAG, SAGA, SAAG-I, which use stochastic averaging scheme, i.e., maintain full gradient from previous iterations, can't be used with random sampling of blocks and mini-batches. A subproblem is formulated by taking selected block of coordinates as variables and fixing the values of rest coordinates, over the selected mini-batch of data points as given below:

$$\min_{w_{v_j}} \quad \frac{1}{|B_i|} \sum_{h \in B_i} L_h(w_{v_j}, w_{/v_j}), \quad j = 1, 2, ..., s, \ i = 1, 2, ..., m, \tag{8}$$

where w_{v_j} denotes block of coordinates v_j and $w_{/v_j}$ denotes coordinates excluding v_j. The size of mini-batches and blocks is selected in such a way to allow iteration complexity that can be handled efficiently by computers on which experiments are performed. On solving

the subproblem, the selected block of coordinates are updated but other coordinates remains unchanged. This runs for predetermined number of epochs n but other exit criteria can be tried depending up on the method used to solve the subproblem and as per requirements. Before this, the idea of mini-batches and block coordinates is studied by few researchers like Wang and Banerjee (2014); Zhao et al. (2014) and Xu and Yin (2015) with Gradient/Proximal Gradient methods, but we project mini-batch and block coordinates setting as an optimization framework (BBOF) independent of any method, as an efficient and scalable framework for solving big data problems. BBOF is a tunable framework and can be tuned to eight different settings by changing the values of m (number of mini-batches) and s (number of blocks), e.g., using Gradient Descent method as a solver in BBOF, following different methods can be tuned:
i) $m = 1$, i.e., one batch contains all data points, $s = 1$, i.e., one block contains all coordinates, then method is GD.
ii) $m = l$, i.e., one batch contains only one data point, $s = 1$, then method is SGD.
iii) $m = 1$, $s = p$, i.e., one block contains only one coordinate, then method is CGD.
iv) $m = B,\ 1 < B < l$, $s = 1$, then method is mini-batch SGD.
v) $m = B,\ 1 < B < l$, $s = p$, then method is mini-batch CGD.
vi) $m = 1$, $s = v,\ 1 < v < p$, then method is BCD (Block Coordinate Gradient Descent).
vii) $m = l$, $s = v,\ 1 < v < p$, then method is stochastic BCD.
viii) $m = B,\ 1 < B < l$, $s = v,\ 1 < v < p$, then method is mini-batch block-coordinate GD (MBGD).

1.3. Brief Literature Review

Stochastic Approximation (SA) approach (Robbins and Monro (1951); Kiefer and Wolfowitz (1952)) used with GD method gives SGD method (Zhang (2004)), which introduces variance in the gradient values because of noisy approximations of gradient. This variance can be reduced by using suitable mini-batches (Li et al. (2014)) instead of using one data point, or by using some variance reduction techniques, like, SAG (Schmidt et al. (2016)), SAGA (Defazio et al. (2014)), SVRG (Johnson and Zhang (2013)) and S2GD (Konečný and Richtárik (2013)) etc., which have same convergence as GD but have iteration complexity nearly equal to SGD method. CD approach along with Block Coordinate Descent (BCD) approach have been studied extensively in literature (Tseng (2001); Nesterov (2012); Beck and Tetruashvili (2013); Richtárik and Takáč (2014); Wright (2015); Lu and Xiao (2015); Shi and Liu (2016) etc.). The combination of BCD and mini-batch SA approach with GD/Proximal GD (PGD) method is studied recently in MRBCD (Zhao et al. (2014)), ORBCD (Wang and Banerjee (2014)) and BSG (Xu and Yin (2015)), which are specific cases of BBOF with GD/PGD method, used to solve different problems. MRBCD and ORBCD have similar idea and theoretical results. BSG solves for convex and non-convex problems and MRBCD solves only convex problems. Both the methods use variance reduction techniques. MRBCD and BSG use extension of SVRG for variance reduction in mini-batch and block coordinate setting. This paper uses BBOF with GD method, i.e., mini-batch and block coordinate setting with GD as a solver, and proposes two new variance reduction methods (see, Section 2), in addition to extending SAG, SAGA and S2GD to this setting.

1.4. Contributions

The contributions of the paper are summarized below:

- Mini-batch and block-coordinate setting, i.e., a combination of Stochastic Approximation (SA) and Coordinate Descent (CD) approaches, have been projected as an efficient and scalable optimization framework, namely, Batch Block Optimization Framework (BBOF), independent of any solver, for solving big data problems. BBOF is a tunable framework and can generate eight different methods by changing m (number of mini-batches) and s (number of blocks) values (see, Sub-section 1.2).
- Two novel methods, namely, SAAG-I and SAAG-II (see, Section 2) have been proposed as a variance reduction methods, under mini-batch and block coordinate setting with GD method and used to solve strongly convex and smooth problem. SAAGs can be used in mini-batched stochastic gradient setting also. Theoretical analysis proves linear convergence for SAAG-II method.
- Variance reduction methods SAG, SAGA, SVRG and S2GD have been extended to mini-batch and block coordinate setting, and compared with the proposed methods.

2. Stochastic Average Adjusted Gradient (SAAG) Methods

SAG, SAGA, SVRG, S2GD and proposed SAAG methods are variance reduction techniques for stochastic gradient. It is very interesting to note that they have very small differences in their update rules, as it is clear from their equations given in this section. All these methods calculate one partial gradient over selected mini-batch at latest iterate which gives the latest gradient values, i.e., step direction to move, calculate one partial gradient over selected mini-batch at old iterate and one partial gradient over whole dataset calculated at old iterates which might not give correct step direction to move. Old gradient values help in reducing the variance in expectation but might lead in wrong direction. Our intuition behind proposing SAAGs is to give more weightage to latest gradient values to get better step direction. So we have divided the latest value of gradient using mini-batch size but old values using number of data points. SAAGs are useful for sufficiently large mini-batches. Our intuitions are followed by the empirical results which clearly show the out-performance of SAAGs over other methods.

In this section, two methods are proposed with two training algorithms, namely, SAAG-I and SAAG-II, both of which use the mini-batch and block-coordinate setting, i.e., BBOF framework. First method is SAAG-I, given by eq.(10) and presented by Algorithm 2, which takes m, s and n as input and sets initial solution (w^0) and total gradient ($\mathbb{G}$) to zero vectors. It randomly selects one mini-batch of data points without replacement and cyclically selects one block of coordinates as in BBOF and formulates a reduced subproblem given by eq.(8) which is solved using eq.(9). The iteration complexity is very low since it calculates two partial gradients w.r.t. selected block of variables over the selected mini-batch of data points and reduces variance by using averaged gradient values calculated using the previous iterations. The parameters are updated for only selected block of coordinates and keeping the rest coordinates unchanged. SAAG-II method is given by eq.(11) and presented

Algorithm 2 SAAG-I

Inputs: $m =$ #mini-batches, $s =$ #blocks, $n =$ #epochs and $\alpha =$ step size.
Initialize: solution $w^0 \in \mathbb{R}^p$ and total gradient $\mathbb{G} \in \mathbb{R}^p$ to zero vector.
for $k = 1, 2, ..., n$ **do**
  Set $u^{k,0} = w^{k-1}$.
  for $i = 1, 2, ..., m$ **do**
    Randomly select one mini-batch B_i without replacement.
    for $j = 1, 2, ..., s$ **do**
      Cyclically select one block of coordinates v_j.
      Formulate a subproblem using block v_j over mini-batch B_i as given in eq.(8).
      Calculate: $g_{B_i,v_j} = \sum_{h \in B_i} \left[L_h^{'} \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j}$ and $\bar{g}_{B_i,v_j} = \sum_{h \in B_i} \left[L_h^{'} \left(u^{k,0} \right) \right]_{v_j}$.
      Update solution as:

$$\begin{aligned} u_{v_j}^{k,i} &= u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{|B_i|} g_{B_i,v_j} - \frac{1}{l} \bar{g}_{B_i,v_j} + \frac{1}{l} \left[\mathbb{G} \right]_{v_j} \right], \\ u_{/v_j}^{k,i} &= u_{/v_j}^{k,i-1}. \end{aligned} \tag{9}$$

      Update total gradient vector, $\left[\mathbb{G} \right]_{v_j} += \frac{1}{l} \left(g_{B_i,v_j} - \bar{g}_{B_i,v_j} \right)$.
    end for
  end for
  $w^k = u^{k,m}$.
end for

Algorithm 3 SAAG-II

Inputs: $m =$ #mini-batches, $s =$ #blocks, $n =$ #epochs and $\alpha =$ step size.
Initialize: solution $w^0 \in \mathbb{R}^p$
for $k = 1, 2, ..., n$ **do**
  Set $u^{k,0} = w^{k-1}$.
  Calculate full gradient, $\mathbb{G}$.
  for $i = 1, 2, ..., m$ **do**
    Randomly select one mini-batch B_i without replacement.
    for $j = 1, 2, ..., s$ **do**
      Cyclically select one block of coordinates v_j.
      Formulate a subproblem using block v_j over mini-batch B_i as given in eq.(8).
      Calculate: $g_{B_i,v_j} = \sum_{h \in B_i} \left[L_h^{'} \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j}$ and $\bar{g}_{B_i,v_j} = \sum_{h \in B_i} \left[L_h^{'} \left(u^{k,0} \right) \right]_{v_j}$.
      Update solution using eq.(9).
    end for
  end for
  $w^k = u^{k,m}$.
end for

by Algorithm 3. SAAG-II algorithm is similar to SAAG-I and uses BBOF framework to provide an efficient and scalable solution to solve the big data problems. Similar to later, it calculates two partial gradients over the mini-batch of data points and uses one gradient over all data points which is calculated at the start of each epoch for variance reduction. Thus, the difference between SAAG-I and SAAG-II algorithms is only in the value of average gradient calculation, former maintains the value from previous iterations starting with zero vector unlike the later which calculates the full gradient at the start of each epoch.

SAG, SAGA, SVRG and S2GD are well known variance reduction methods for SGD setting. SVRG has been extended to mini-batch and block-coordinate setting, for variance reduction, in MRBCD (Zhao et al. (2014)) and BSG (Xu and Yin (2015)). In this paper, SAG, SAGA and S2GD are also extended to this setting for comparing against SAAG methods. To closely study, all these methods in the mini-batch and block-coordinate setting, the parameter update rules are given by equations (10-15), where $u_{v_j}^{k,i}$ denotes the k^{th} parameter update rule for v_j block of coordinates over the i^{th} mini-batch B_i, $u_{<v_j}^{k,i}$ denotes variables of $u^{k,i}$ which are already computed and $u_{\geq v_j}^{k,i-1}$ denotes variables of $u^{k,i-1}$ that have not yet to be advanced to iteration (k,i) along with v_j block of variables being updated. As it is clear from following equations, SAAG-I, like, SAG and SAGA, maintains the full gradient from previous iterations by saving the partial gradients over the mini-batches but SAAG-II, like, S2GD and SVRG, calculates the full gradient at the start of the epochs and uses inside the inner loops to reduce the variance in gradient value. S2GD and SVRG have similar parameter update rules, although different learning algorithms, so only SVRG is given.

SAAG-I:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} - \frac{1}{l} \sum_{h \in B_i} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} + \frac{1}{l} \sum_{h=1}^{l} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} \right] \tag{10}$$

SAAG-II:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} - \frac{1}{l} \sum_{h \in B_i} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} + \frac{1}{l} \sum_{h=1}^{l} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} \right] \tag{11}$$

SAGA:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} - \frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} + \frac{1}{l} \sum_{h=1}^{l} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} \right] \tag{12}$$

SAG:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{l} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} - \frac{1}{l} \sum_{h \in B_i} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} + \frac{1}{l} \sum_{h=1}^{l} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} \right] \tag{13}$$

SVRG:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \alpha \left[\frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} - \frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} + \frac{1}{l} \sum_{h=1}^{l} \left[L_h' \left(u^{k,0} \right) \right]_{v_j} \right] \tag{14}$$

MBGD:

$$u_{v_j}^{k,i} = u_{v_j}^{k,i-1} - \frac{\alpha}{|B_i|} \sum_{h \in B_i} \left[L_h' \left(u_{<v_j}^{k,i}, u_{\geq v_j}^{k,i-1} \right) \right]_{v_j} \tag{15}$$

Complexity Analysis The per-iteration complexity for all the methods is $O(Bv)$ (assuming constant mini-batch size B and block size v), which is controllable as per the machine capability, unlike, complexity $O(lp)$ of GD method which might be infeasible to process. Using same algorithm structure, as given in Algorithm 1, per epoch component gradient evaluations for S2GD, SVRG and SAAG-II are l (calculation of full gradient at start of epoch) + $2l$ ($2Bm$, as $Bm = l$), but for SAG, SAGA and SAAG-I are $2l$ since they maintain full gradient from previous iterations and don't calculate at start of epoch. Technically, later case take only l gradient evaluations per epoch at the expense of saving gradients which need $O(lp)$ extra memory in general but only $O(l)$ memory to solve problem given by eq. (3).

3. Analysis

Theorem 1 *Suppose for objective function given by eq. (1), under the assumptions of μ-strong convexity, component-wise Lipschitz continuity of gradient, constant step size α random sampling without replacement for blocks and random sampling with replacement for mini-batches SAAG-II, converges linearly to optimal value p^*, for constant mini-batch size B, constant block size v, as given below,*

$$\mathbb{E}\left[f\left(w^k\right) - p^*\right] \leq \left(1 - \frac{2v\alpha\mu}{p}\right)^k \left(f\left(w^0\right) - p^*\right) + \frac{pR_0^2}{2v\mu}\left(L\alpha\left(3 + \frac{2B^2}{l^2}\right) - 1 + \frac{B}{l}\right) \tag{16}$$

$$\textit{where } \left\| \frac{1}{|B_i|} \sum_{h \in B_i} \left[L_h'(w) \right]_{v_j} e_j \right\| \leq R_0, \ \forall\ w, i, j, B_i, v_j \ \textit{and}\ e_j(i) = \begin{cases} 1 \ \textit{if}\ i \in v_j \\ 0 \ \textit{else} \end{cases} .$$

Detailed derivation of convergence is given in Appendix A (supplementary material).

4. Numerical Experiments

In this section, experimental results[1] are provided and proposed methods, SAAG-I and SAAG-II, are compared against the variance reduction techniques, like, BSG (Xu and Yin (2015)), MRBCD (Zhao et al. (2014)) and extensions of SAG (Schmidt et al. (2016)), SAGA (Defazio et al. (2014)), SVRG (Johnson and Zhang (2013)) and S2GD (Konečný and Richtárik (2013)) to mini-batch and block-coordinate setting. MBGD (Mini-batch Block-coordinate Gradient Descent) which is simple method without any variance reduction is also considered as presented in eq.(15). In this experimentation, logistic regression problem, given in eq.(3) is solved. Since BSG and MRBCD use SVRG for variance reduction, thus under our experimental settings BSG, MRBCD and extension of SVRG to mini-batch and block-coordinate setting become same so only SVRG is mentioned in the

1. Experimental results can be reproduced using code available at link: https://sites.google.com/site/jmdvinodjmd/code/saag

experiments. All the methods use the same algorithmic structure as given in eq. (1) with change only in update rule (variance reduction method) at step 9. Experiments are conducted with datasets as given in Table 1. All the methods use averaged gradient value for variance reduction, calculated either at start of epochs or maintained from previous iterations. To store the full gradient only a vector of length l is used since it stores only constant value for each data point instead of storing the complete gradient vector which will take $l * p$ memory space. Step size is calculated using two different methods, namely, back tracking line search method and constant step size determined using Lipschitz constant. It is interesting to note that in backtracking line search using all data points for big data problems can be very expensive so backtracking line search is performed only on selected mini-batch with parameter values as $\beta = 0.5$ (to decrease the step size) and $\gamma = 0.1$ (i.e., allow 10% decrease of objective function) for all methods. With Lipschitz constant L, step size is taken as $1/L$ for all the methods. This is to be noted that all experiments have been conducted on *single node, MacBook Air (8 GB 1600 MHz DDR3, 1.6 GHz Intel Core i5, 256 SSD)*. For all methods, regularization parameter is set as $C = 1/l$.

Table 1: Datasets[2] used in experimentation

Dataset	#classes	#features	#datapoints
news20.binary	2	1,355,191	19,996
rcv1.binary	2	20,242	47,236
real-sim	2	72,309	20,958
webspam	2	254	350,000
SUSY	2	18	5,000,000

Following figures presents the experimental results with bench marked datasets given in Table 1, with different mini-batch and block sizes. Figures plot the difference of objective function and optimum value against the number of epochs. Following observations can be made from the experimental results:
(i) In general, SAAG methods, i.e., SAAG-I and SAAG-II, converges faster than other solvers with SAAG-II outperforms all other methods.
(ii) In general, with increase in size of the mini-batch and block, SAAG methods perform better as compared with rest of the solvers, but for smaller sizes, e.g., with mini-batch of one data point they don't perform.
(iii) Results with line search method and using Lipschitz constant to determine the step size give similar results and prove the superiority of SAAG methods over others.
(iv) As expected from equations of different solvers, when block size is equal to #features and mini-batch size is equal to #data points, i.e., in GD method setting all methods converge exactly same (due to space constraints figure is not included).
(v) Plots of sub-optimality against time shows similar results as shown for sub-optimality against epochs (see supplementary material for results).
(vi) Line search results are smoother than constant step size. Moreover, line search results are better than constant step for SUSY but for other datasets, constant step size gives better

2. All used datasets are available at link: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

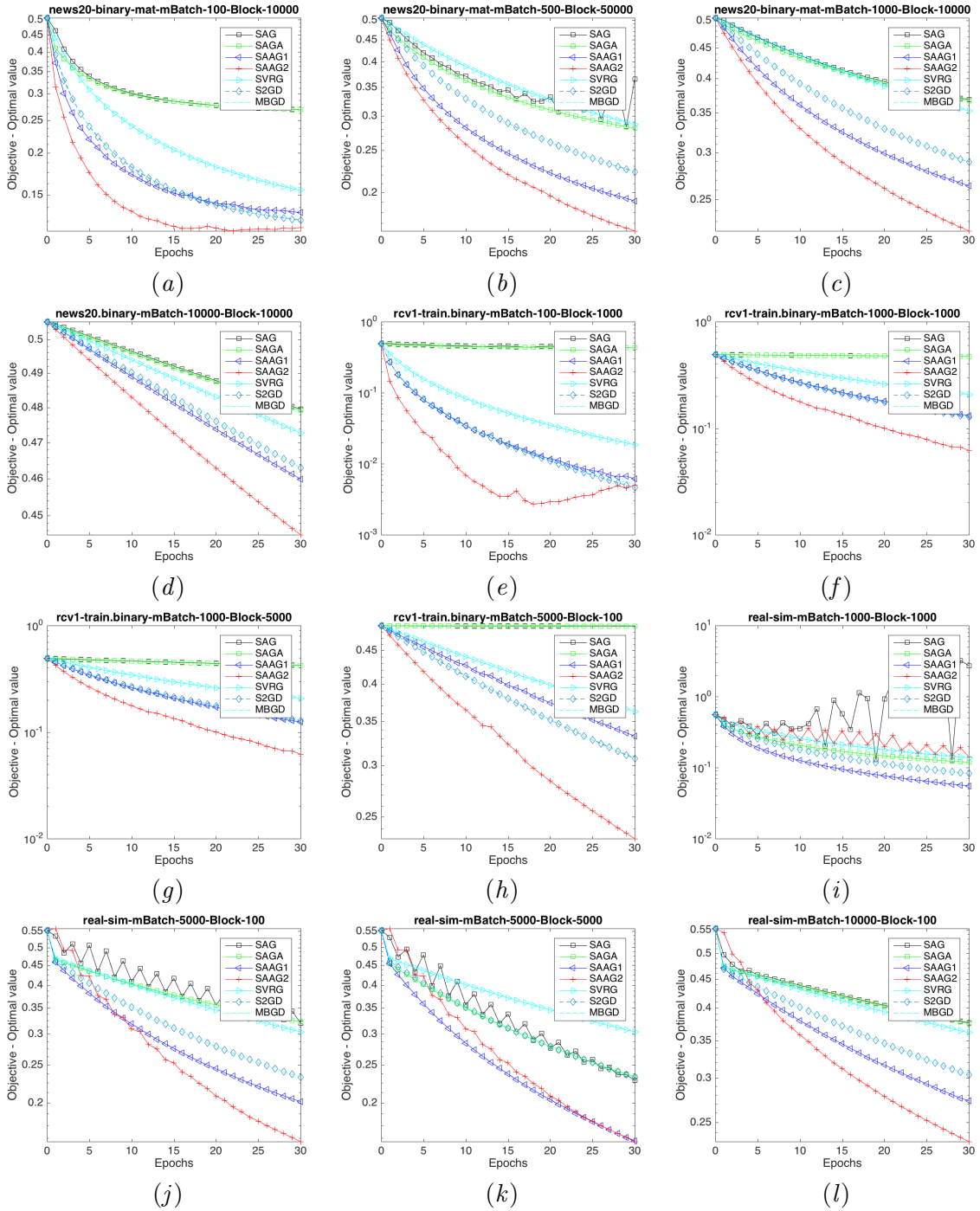

Figure 1: Results with constant step size using Lipschitz constant (news20 - (a), (b), (c), (d); rcv1 - (e), (f) (g), (h); real-sim - (i), (j), (k), (l)).

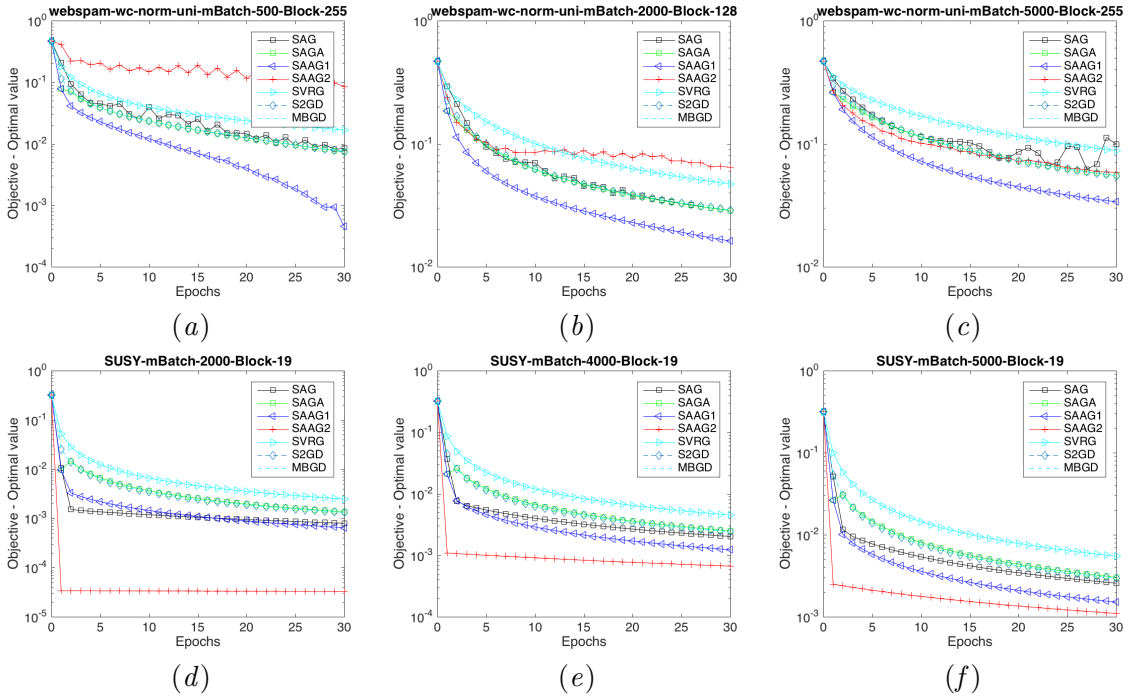

Figure 2: Results with constant step size using Lipschitz constant (webspam - (a), (b), (c); SUSY - (d), (e), (f)).

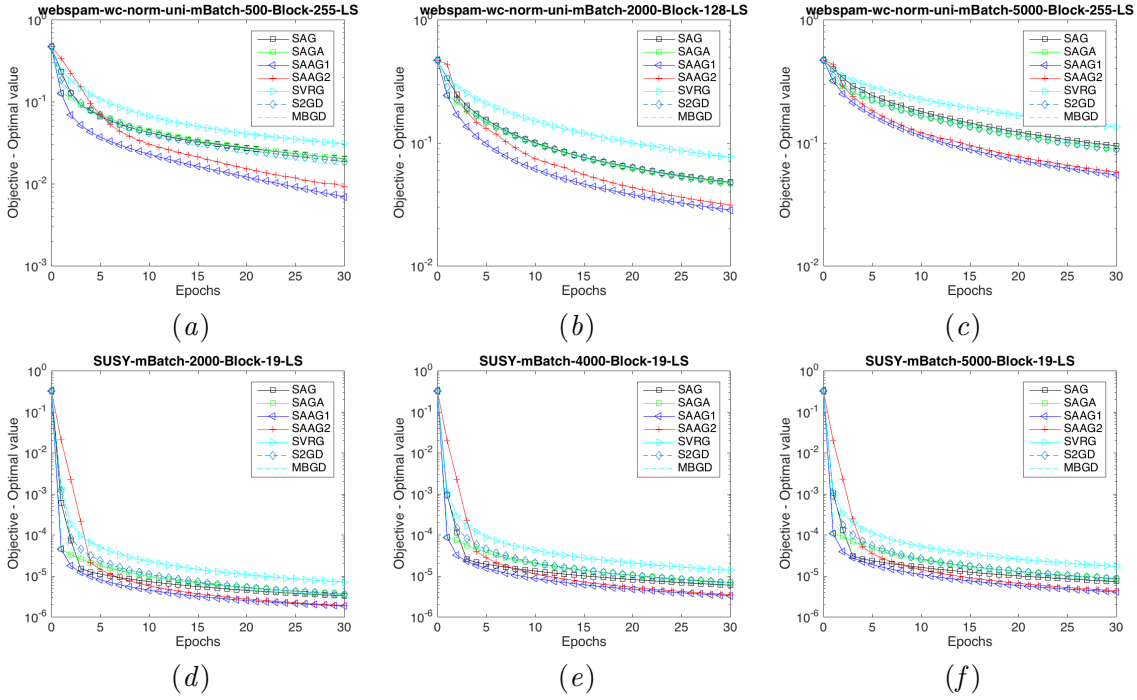

Figure 3: Results with line search method (webspam - (a), (b), (c); SUSY - (d), (e), (f)).

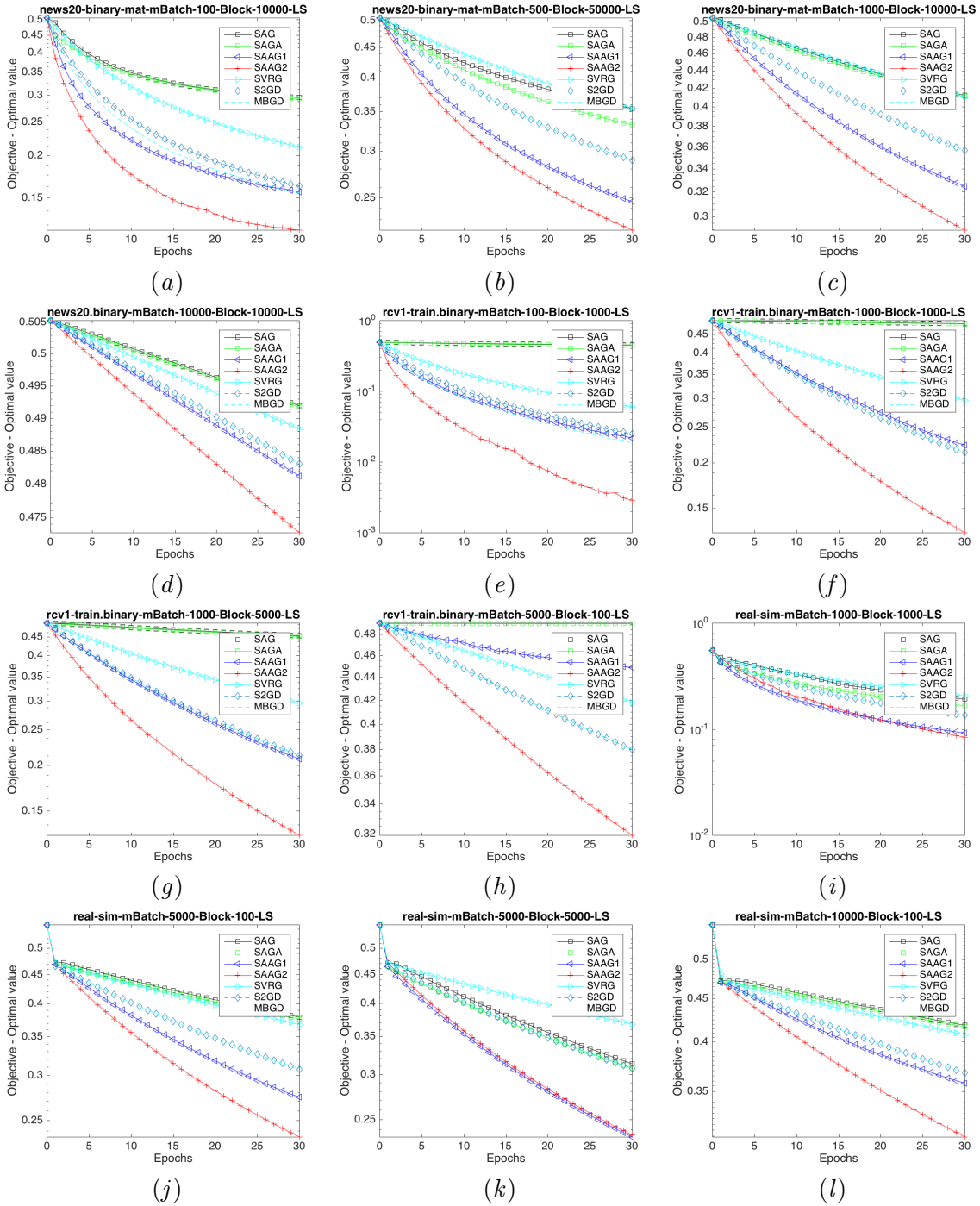

Figure 4: Results with line search method (news20 - (a), (b), (c), (d); rcv1 - (e), (f) (g), (h); real-sim - (i), (j), (k), (l)).

results. Major reason for this is the larger step size obtained using Lipschitz constant which is sometimes even greater than one but for line search maximum size is one which keeps on decreasing and other possible reason is that the line search is performed on selected mini-batch and not on full dataset.
(vii) SUSY has only 18 features so block size has been taken equal to number of features, moreover for webspam all features has been taken with mini-batches of 500, 200 and 5000. These experiments form mini-batched stochastic gradient setting and prove the effectiveness of SAAGs for this setting also, and algorithms and formulas can be obtained for this setting by setting $v = p$.

5. Conclusion and Future Scope

In this paper, Mini-batch and block-coordinate setting, i.e., a combination of SA and CD approaches, has been projected as an efficient and scalable optimization framework, namely, BBOF, independent of any solver, for solving big data problems. BBOF has been discussed with Gradient Descent as a solver and two novel variance reduction methods (SAAG-I and SAAG-II) have been proposed and used to solve strongly convex and smooth problems. Theoretical results prove linear convergence and empirical results with bench marked datasets prove faster convergence for the proposed methods than existing methods of variance reduction. BBOF and SAAG can be extended to parallel and distributed settings, and to solve other convex and non-convex problems. In addition, SAAG methods are further open for analyzing the convergence for SAAG-I and SAAG-II algorithms, and their theoretical properties.

Acknowledgments

First author would like to thank Ministry of Human Resource Development, Government of INDIA, for providing fellowship (University Grants Commission - Junior Research Fellowship) to pursue his Ph.D. work.

References

Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 1646–1654, Cambridge, MA, USA, 2014. MIT Press.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 315–323. Curran Associates, Inc., 2013.

J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23:462–466, 1952.

Jakub Konečný and Peter Richtárik. Semi-Stochastic Gradient Descent Methods. 1:19, 2013. URL http://arxiv.org/abs/1312.1666.

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*, KDD '14, pages 661–670, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9.

Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Math. Program.*, 152(1):615–642, 2015. ISSN 1436-4646.

Yu. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Math. Program.*, 144(1-2):1–38, 2014.

Herber Robbins and Sutton Monro. A stochastic approximation method. vol-22:pp. 400–407, 1951.

Mark Schmidt. Convergence rate of stochastic gradient with constant step size. Technical report, 2014.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Math. Program.*, pages 1–30, 2016.

Ziqiang Shi and Rujie Liu. A better convergence analysis of the block coordinate descent method for large scale machine learning. aug 2016. URL http://arxiv.org/abs/1608.04826.

P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optim. Theory Appl.*, 109(3):475–494, June 2001. ISSN 0022-3239.

Huahua Wang and Arindam Banerjee. Randomized Block Coordinate Descent for Online and Stochastic Optimization. *arXiv preprint*, (1):1–19, 2014.

Stephen J. Wright. Coordinate descent algorithms. *Math. Program.*, 151(1):3–34, June 2015. ISSN 0025-5610.

Yangyang Xu and Wotao Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5.

Tuo Zhao, Mo Yu, Yiming Wang, Raman Arora, and Han Liu. Accelerated Mini-batch Randomized Block Coordinate Descent Method. *Advances in Neural Information Processing Systems*, pages 3329–3337, 2014.