Comparative Evaluation of Variants of Stochastic Gradient Descent for Multilabel Classification Loss Functions

Gaurush Hiranandani[1], Shrey Pareek[2]

***Abstract*— In this paper, we study the relative performance of nine recent variants of the stochastic gradient descent (SGD) algorithm across three data sets for three loss functions accustomed to a multi-label-classification setting. We present a theoretical analysis of SGD and two loss functions and highlight a critical drawback in the multi-label-classification literature. We also develop a set of guidelines for readers on the recommended use of these methods for different cases.**

MODIFICATIONS SINCE MILESTONE

In this section, we briefly highlight the differences in the final iteration of the project and the project milestone.

A. *Theoretical Analysis*

We have included a theoretical analysis relating to the convergence of stochastic gradient descent and some critical analysis of the Hamming and Subset Loss settings.

B. *Additional Dataset*

Per the feedback from Dr. Sun following the project proposal, a suggestion was made to check the performance of the algorithms on a larger dataset. Therefore, in addition to the original music-emotions dataset, we have included a much larger MediaMill data set available at KEEL.[1]

C. *Diminishing Step Size*

As stated in the original proposal, we have implemented a diminishing step size rule for the algorithms in the final report. In the previous iterations of the project, we presented the results with a constant step size rule.

I. INTRODUCTION

Recent advances in the computational power of C.P.U.'s and G.P.U.'s along with the easier accessibility to powerful cloud computing has led to the advent of large scale machine learning problems. The ability to solve these problems relies not only on the type of machine learning algorithm or underlying loss function but also on the underlying optimization routine. Owing to the large scale of these problems, methods that can effectively decompose these larger datasets into smaller subproblems have gained prominent attention in the recent past. SGD [1] is one such method, which has shown promise in addressing big data challenges. Despite their popularity and efficacy, there exists a considerable gap in the understanding of the behavior of these methods under different conditions.

In an attempt to bridge this knowledge gap, we seek to explore and understand the performance of nine variants of SGD – three variants of Stochastic Variance-Reduced Gradient (SVRG) [2], Stochastic Average Gradient (SAG) [3], SAGA [4], Semi-Stochastic Gradient Descent (s2gd) [5], VR-Lite [6], and Batching and Mixing SVRG [7].

In this paper, we explore the relative performance of these algorithms on three loss functions viz. Hamming Loss [8], Subset 0/1 Loss [8], and Gebru Loss [9]. The analysis is performed using two multi-label classification (MLC) datasets provided by [10] and [11]. The aim of this paper is to provide a deeper insight into the performance differences of these methods and the underlying intuition behind them.

Problem Scope - It should be noted that the goal of this study is not to achieve the best classification accuracy for MLC. Instead, we are only interested in the relative performance of these algorithms under similar experimental settings. We use the same parameters across all settings in order to provide a fair comparison of these algorithms, . We do not attempt to fine tune the hyper-parameters. Under some circumstances, the assumptions may generate contradictory or unreliable results. However, the identification and resolution of these anomalies is beyond the scope of this paper.

Before delving into the experimental aspects of the study, we provide a theoretical introduction and analysis on the large scale optimization methods, MLC, and the loss functions being considered. The paper is organized as follows. We provide the problem formulation and notations in Section II. We discuss SGD and it's variants in Section III and IV, respectively. We provide a discussion on MLC and the loss function in Sections V and VI. The experimental analysis and results are presented in Sections VII and VIII. We conclude in Section IX.

II. PROBLEM DEFINITION AND NOTATIONS

We define our problem in terms of a MLC problem. Our goal is to determine a prediction function $h : \mathcal{X} \to \mathcal{Y} \in \{0, 1\}^M$ on an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ such that, given $x \in \mathcal{X}$, the value $h(x)$ offers an accurate prediction about the true output y. To do this, one should choose the prediction function h by attempting to minimize a risk measure over an adequately selected family of prediction functions, call it $\mathcal{H}$. We sometimes abuse notation and treat h to be a vector valued prediction function, depending on

[1]Gaurush Hiranandani is with the Department of Computer Science, University of Illinois at Urbana-Champaign, IL-61820 gaurush2@illinois.edu

[2]Shrey Pareek is with the Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, IL-61820 spareek2@illinois.edu

[1]http://sci2s.ugr.es/keel/dataset_smja.php?cod=922#sub1

the context. Similarly, we denote x and y as vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, depending on the context.

We assume that the prediction function h has a fixed form and is parameterized by a real vector $w \in \mathbb{R}^d$ over which the optimization is to be performed. In this project, we take the family of linear functions, that is, our prediction function takes the form $w^T x$. In general, for some given $h(\cdot;\cdot) : \mathbb{R}^{d_x} \times \mathbb{R}^d \to \mathbb{R}^{d_y}$, we consider the family of prediction functions

$$\mathcal{H} := \{h(\cdot; w) : w \in \mathbb{R}^d\}$$

We aim to find the prediction function in this family which minimizes certain losses incurred from inaccurate predictions. We assume a loss function denoted by $\ell : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to \mathbb{R}$ as one that, given an input-output pair (x, y), yields the loss $\ell(h(x; w); y)$ when $h(x; w)$ and y are the predicted and true outputs, respectively.

The parameter vector w is chosen to minimize the expected loss that would be incurred from any input-output pair. We assume that losses are measured with respect to a probability distribution $P(x, y)$. The objective function we wish to minimize is

$$R(w) = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} \ell(h(x; w); y) dP(x; y) = \mathbb{E}[\ell(h(x; w); y)] \tag{1}$$

We say that $R : \mathbb{R} \to \mathbb{R}$ yields the risk (i.e., expected loss) given a parameter vector w with respect to the probability distribution P.

In reality, P is usually unknown. Thus, in practice, we work with an estimate of the expected risk R. We take the set of $n \in \mathbb{N}$ independently drawn input-output samples $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$, with which one may define the empirical risk function $R_n : \mathbb{R}^d \to \mathbb{R}$ by

$$R_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i; w); y_i) \tag{2}$$

Let us represent a sample (or set of samples) by a random seed ξ. In this entire document, we refer to the loss incurred for a given $(w; \xi)$ as $f(w; \xi)$, i.e.,

f is the composition of the loss function ℓ and the prediction function h.

From the above definition of loss composed with prediction function, we have the expected risk for a given w as:

$$R(w) = E[f(w; \xi)], \tag{3}$$

loss incurred by the parameter vector w with respect to the i^{th} sample as:

$$f_i(w) := f(w; \xi_{[i]}), \tag{4}$$

and a sample estimator of expected risk as:

$$R_n(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{5}$$

We use [i] to denote the i^{th} element of a fixed set of realizations of a random variable, and k to denote the k^{th} element of a sequence of random variables.

III. STOCHASTIC GRADIENT DESCENT [2]

Optimization in Machine Learning refers to the selection of parameters which are optimal w.r.t. a given learning problem defined in the sense of minimizing a loss function. Optimization methods for machine learning fall into two broad categories - *batch* and *stochastic*. Traditional gradient-based methods such as Gradient Decent (GD), involve a batch approach, and have been effective for solving small-scale problems. On the flip side, stochastic methods such as SGD proposed by [12] are suitable for large scale optimization problems. We now describe a brief analysis on SGD and it's inherent challenges.

In SGD, the $(k+1)^{th}$ iterate is defined as:

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f_{i_k}(w_k) \tag{6}$$

Here, the index i_k (corresponding to a random seed $\xi_{[i_k]}$, i.e., the sample-label pair $(x_{i_k}; y_{i_k})$ is chosen randomly from $[1, ..., n]$ (n being the total number of samples) and α_k is a positive step size. While each direction $\nabla f_{i_k}(w_k)$ might not be one of descent from w_k, if it is a descent direction in expectation, then the sequence $\{w_k\}$ can be guided toward a function minimizer. The basic algorithm is described as -

Algorithm 1 SGD Algorithm

1: Choose an initial iterate w_0.
2: **for** $k = 1, 2, ...$ **do**
3: Generate a realization of the random variable ξ_k.
4: Compute a stochastic vector $g(w_k, \xi_k)$.
5: Choose a step size $\alpha_k > 0$.
6: Set the new iterate as $w_{k+1} \leftarrow w_k - \alpha_k g(w_k, \xi_k)$.

A. *Variance Reduction Analysis for SGD*

The theory related to the analysis of stochastic gradient descent requires us to make some assumptions. The first assumption is as follows:

Assumption 1 (Lipschitz-continuous objective gradients): The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the gradient function of F, namely, $\nabla F : \mathbb{R}^d \to \mathbb{R}^d$, is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.,

$$\|\nabla F(w) - \nabla F(\overline{w})\|_2 \leq L\|w - \overline{w}\|_2 \ \ \forall \ \ \{w, \overline{w}\} \subset \mathbb{R}^d \tag{7}$$

We call such functions as L-smooth functions. This assumption tells us that the gradient of F does not change arbitrarily quickly with respect to the parameter vector. Such an assumption is essential for convergence analyses of most gradient-based methods; without it, the gradient would not provide a good indicator for how far to move to decrease F. Following directly from Assumption 1, we have that

$$F(w) \leq F(\overline{w}) + \nabla F(\overline{w})^T (w - \overline{w}) + \frac{1}{2} L\|w - \overline{w}\|_2^2 \ \forall \ \ \{w, \overline{w}\} \subset \mathbb{R}^d. \tag{8}$$

[2]The following analysis is borrowed from [1] but written succinctly depending on the scope of this project.

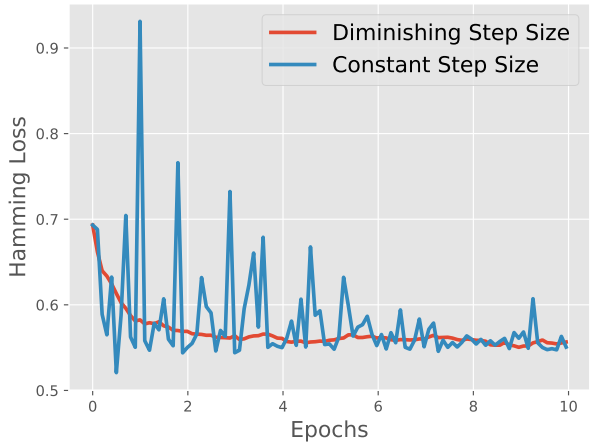

Fig. 1: Comparison of SGD performance under a constant and diminishing step size rule. Performance is same for both the cases. Oscillations are more in constant stepsize in comparison to diminishing step sizes.

(8) gives us an immediate lemma as follows.

Lemma 1: Under Assumption 1, the iterates of SG (Algorithm 1) satisfy the following inequality for all $k \in \mathbb{N}$:

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\alpha_k \nabla F(w_k)^T \mathbb{E}_{\xi_k}[g(w_k, \xi_k)] + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}\|g(w_k, \xi_k)\|_2^2 \quad (9)$$

This lemma shows that, regardless of how SGD arrived at w_k, the expected decrease in the objective function yielded by the kth step is bounded above by a quantity involving: (i) the expected directional derivative of F at w_k along $-g(x_k, \xi_k)$ and (ii) the second moment of $g(x_k, \xi_k)$. For example, if $g(w_k, \xi_k)$ is an unbiased estimate of $\nabla F(w_k)$, then it follows from Lemma 1 that -

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2]. \quad (10)$$

In the following analysis, it is shown that convergence of SGD is guaranteed as long as the stochastic directions and step sizes are chosen such that the right-hand side of (9) is bounded above by a deterministic quantity that asymptotically ensures sufficient descent in F. This is ensured by adding more constraints on the first and second moments of the stochastic directions $\{g(w_k, \xi_k)\}$. Therefore, in order to limit the harmful effect of the right most term in (10), we restrict the variance of $g(w_k, \xi_k)$, i.e.,

$$\mathbb{V}_{\xi_k}[g(w_k, \xi_k)] := \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] - \|\mathbb{E}_{\xi_k}[g(w_k, \xi_k)]\|_2^2. \quad (11)$$

Once the above variance is bounded, we can control the second term on the right side of (10) by controlling the step size. The most intuitive solution to counter this challenge is to use a diminishing step size law such as $\alpha_d = \frac{1}{k^\beta}$. Here k is the iteration number and $0.5 < \beta \leq 1$.

As a vanilla example, we compare the performance of SGD under the two step size rules in Fig.1. As is evident, use of a diminishing step size reduces the amplitude of oscillations, even though the solution converges to approximately the same solution. Thus, use of a diminishing step size leads to a stable convergence, but does not improve the overall solution quality. It is here that the need for alternative means of variance and noise reduction methods is highlighted. In the following section we describe gradient aggregation-based SGD variants that claim to resolve the issues arising due to noisy gradients.

IV. SGD VARIANTS

Optimization algorithms have information regarding the gradient estimates of previously visited training samples. Gradient aggregation methods make use of this information and improve upon the lower bound [13] for optimization. These algorithms enjoy linear convergence with low computing times in practice and are briefly described here.

A. *Stochastic Variance Reduce Gradient (SVRG) - Three Updates*

Variance reduction is employed so as to guarantee linear convergence or better for SGD without the use of a diminishing step size. In order to reduce variance, SVRG [2] uses an estimated $\tilde{w}$ close to the optimal w stored at every iteration and the average gradient $\tilde{\mu}$ is computed. At every *epoch* of the dataset, $\tilde{w}$ and $\tilde{\mu}$ are updated and used for the next iteration. By employing an update of the form of $w_{(k)} = w_{(k-1)} - \alpha(\nabla f_i(w_{(k-1)}) - \nabla f_i(\tilde{w}) + \tilde{\mu})$, where α is the step size and i is randomly sampled from $\{1, ..., n\}$, the variance is explicitly reduced. The details of the algorithm are provided in - Algorithm 2

SVRG introduces three types of update rules at every iteration ((a), (b), and (c) in Algorithm 2). SVRG Update (a) picks the most recent $\tilde{w}_i$ as the update to w. SVRG Update (b) averages the $\tilde{w}_i$ over each epoch. SVRG Update (c) randomly samples a $\tilde{w}_i$ to use as an update to w.

The authors of [2] have shown that the algorithm has linear convergence assuming that the objective functions f_i are smooth and convex, and the overall objective function is strongly convex.

B. *SAGA*

SAGA is inspired by SVRG. Instead of approximating the parameters w, SAGA stores the gradient vectors for each sample i. Let $u_i(w_k)$ be the gradient vector for sample i with the weight vector w_k. At each iteration, a random sample j is picked and the weight vector is updated as $w_k = w_{(k-1)} - \alpha\left[\nabla f_j(w_{(k-1)} - u_j + \frac{1}{n}\sum_{i=1}^{n} u_i\right]$. Furthermore, the gradient of sample j is updated as $u_j = \nabla f_j(w_{(k-1)})$. See Algorithm 3 for more details.

Through storing the full gradient vectors, each iteration corrects the new gradient by using the mean of the gradients through-out the full sample. The authors of [4] have shown that SAGA has a linear convergence rate similar to that of SVRG with slightly better constants for a specifically chosen step size.

Algorithm 2 SVRG Methods

Choose an initial iterate $w_1 \in \mathbb{R}_{\mathbb{d}}$, stepsize $\alpha > 0$, and positive integer m.
for $k = 1, 2, \ldots$ **do**
  Compute the batch gradient $\nabla R_n(w_k)$.
  Initialize $\tilde{w}_1 \leftarrow w_k$.
  for $j = 1, \ldots, m$ **do**
    Chose i_j uniformly from $\{1, \ldots, n\}$.
    Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_n(w_k))$.
    Set $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$.
  Option (1): Set $w_{k+1} = \tilde{w}_{m+1}$
  Option (2): Set $w_{k+1} = \frac{1}{m} \sum_{j=1}^{m} \tilde{w}_{j+1}$
  Option (3): Choose j uniformly from $\{1, \ldots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.

Algorithm 3 SAGA Method

Choose an initial iterate $w_1 \in \mathbb{R}^d$ and stepsize $\alpha > 0$.
for $i = 1, \ldots, n$ **do**
  Compute $\nabla f_i(w_1)$.
  Store $\nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1)$.
for $k = 1, 2, \ldots$ **do**
  Choose j uniformly in $\{1, \ldots, n\}$.
  Compute $\nabla f_j(w_k)$.
  Set $g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]})$.
  Store $\nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k)$.
  Set $w_{k+1} \leftarrow w_k - \alpha g_k$.

C. *Stochastic Average Gradient (SAG)*

SAG [3] is the earliest proposed variant of gradient aggregation algorithms we have surveyed. It incorporates a memory of previous gradients which allows the algorithm to achieve linear convergence rates with fixed step size. It claims to outperform the standard SGD algorithm.

Under a strongly convex sum of smooth functions, the SAG algorithm has linear convergence. The difference between SAG and SAGA is that SAG updates the weight vectors as $w_k = w_{(k-1)} - \alpha \frac{1}{n} \left[\nabla f_j(w_{(k-1)} - u_j + \sum_{i=1}^{n} u_i \right]$. The disadvantage of SAG is that it uses a biased estimator of the gradient unlike SAGA as detailed by [4] (see Algorithm 4).

Algorithm 4 SAG Method

Choose an initial iterate $w_1 \in \mathbb{R}^d$ and stepsize $\alpha > 0$.
for $i = 1, \ldots, n$ **do**
  Compute $\nabla f_i(w_1)$.
  Store $\nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1)$.
for $k = 1, 2, \ldots$ **do**
  Choose j uniformly in $\{1, \ldots, n\}$.
  Compute $\nabla f_j(w_k)$.
  Set $g_k \leftarrow \frac{1}{n} \{ \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \sum_{i=1}^{n} \nabla f_i(w_{[i]}) \}$.
  Store $\nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k)$.
  Set $w_{k+1} \leftarrow w_k - \alpha g_k$.

D. *Semi-Stochastic Gradient Descent - S2GD*

One of the SVRG [2], update methods picks w uniformly at random. S2GD imposes a distribution on the update instead of a uniform distribution. The recent weight updates get higher probability mass than the older one. Thus, SVRG then is a special case of S2GD (see Algorithm 5).

The convergence rate of S2GD is the same as that of SVRG with differences in constants. Both exhibit linear convergence. However, [5] showed that S2GD+, a variant of S2GD without theoretical analysis, shows better convergence in practice.

Algorithm 5 S2GD Method

Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0, \nu > 0, \beta > 0$, and positive integer m.
for $k = 1, 2, \ldots$ **do**
  Compute the batch gradient $\nabla R_n(w_k)$.
  Initialize $\tilde{w}_1 \leftarrow w_k$.
  for $j = 1, \ldots, m$ **do**
    Chose i_j uniformly from $\{1, \ldots, n\}$.
    Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_n(w_k))$.
    Set $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$.
  Choose j from $\{1, \ldots, m\}$ with probability $\frac{1}{\beta}(1 - \nu\alpha)^{m-t}$ for $t \in [m]$.
  Set $w_{k+1} = \tilde{w}_{j+1}$.

E. *VR-Lite*

VR-Lite [6] is proposed as a variance reduced SGD variant to reduce the high memory usage in large batch gradient computations methods such as in S2GD. VR-Lite performs SGD for the first effective pass of the data to initialize the w vectors and the iteration averages $\bar{w}$ and $\bar{g}$. VR-Lite then updates w by going through the samples without replacement and updating the parameters by using $\bar{w}$ and $\bar{g}$ as correction terms (see Algorithm 6).

VR-Lite preserves linear convergence rates under strongly convex functions. However, the author noted that although convergence can be shown experimentally, proving it theoretically is difficult.

F. *Batching and Mixed SVRG*

Batching SVRG [7] (Algorithm 7) differs from SVRG in the sense that instead of sampling over the entire dataset, a batch is chosen at every iteration. The samples are then drawn without replacement from the batch. Each update is then similar to that of SVRG where the gradient of the entire batch is added as a correction term. The batch sizes can be dynamically sized from iteration to iteration.

Mixed SVRG [7] (Algorithm 8) is a hybrid between SGD and Batching SVRG. The only difference between Batching SVRG and Mixed SVRG is during the sampling. If the sample chosen is not in the batch, then normal stochastic gradient is calculated and used as an update. If the chosen sample is in the batch, then an SVRG update is done.

Algorithm 6 VR-Lite Method

Choose initial iterates $w \in \mathbb{R}^d$ and stepsize $\alpha > 0$.
Initialize $\bar{w} = w$ and $\bar{g} = \nabla f(w)$.
for $i = 1, \ldots, n$ **do**
  Sample i uniformly in $\{1, ..., n\}$.
  Compute $\nabla f_i(w)$.
  Set $w \leftarrow w - \alpha \nabla f_i(w)$.
  Set $\bar{w} = \bar{w} + w$.
  Set $\bar{g} = \bar{g} + \nabla f_i(w)$.
Set $\bar{w} = \frac{1}{n}\bar{w}$.
Set $\bar{g} = \frac{1}{n}\bar{g}$.
for $k = 1, 2, \ldots$ **do**
  Initialize $\tilde{w} = \tilde{g} = 0$.
  for $j = 1, \ldots, n$ **do**
    Chose i_j uniformly from $\{1, \ldots, n\}$ without replacement.
    Set $w \leftarrow w - \alpha(\nabla f_{i_j}(w) - \nabla f_{i_j}(\bar{w}) + \bar{g})$.
    Set $\tilde{w} = \tilde{w} + w$.
    Set $\tilde{g} = \tilde{g} + \nabla f_{i_j}(w)$.
  Set $\bar{w} = \frac{1}{n}\tilde{w}$.
  Set $\bar{g} = \frac{1}{n}\tilde{g}$.

Algorithm 7 Batching SVRG Method

Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0$, and positive integers B and m.
for $k = 1, 2, \ldots$ **do**
  Sample a batch of size B without replacement from $\{1, ..., n\}$.
  Compute the batch gradient $\nabla R_B(w_k) = \sum_{i \in B} \nabla f_i(w_k)$.
  Initialize $\tilde{w}_1 \leftarrow w_k$.
  for $j = 1, \ldots, m$ **do**
    Chose i_j uniformly from $\{1, \ldots, n\}$.
    Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_B(w_k))$.
    Set $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$.
  Choose j uniformly from $\{1, \ldots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.

The authors of [7] show proof for linear convergence rates for Batching and Mixed SVRG for strongly convex functions. For Mixed SVRG, a diminishing step size is still required for the stochastic gradient part just like SGD to guarantee the linear convergence.

V. MULTI-LABEL-CLASSIFICATION

Let $\mathcal{L} = \lambda_1, \lambda_2, ..., \lambda_m$ be a finite set of class labels. We assume that an instance $\mathbf{x} \in \mathcal{X}$ is (non-deterministically) associated with a subset of labels $L \in 2^{\mathcal{L}}$; this subset is often called the set of relevant labels, while the complement $L \setminus \mathcal{L}$ is considered as irrelevant for $\mathbf{x}$. We identify a set L of relevant labels with a binary vector $\mathbf{y} = (y_1, y_2, ..., y_m)$, in which $y_i = 1 \iff \lambda_i \in L$. By $\mathcal{Y} = \{0, 1\}^m$ we denote the set of possible labellings. We denote by $P(\mathbf{y}|\mathbf{x})$ the conditional distribution of $\mathbf{Y} = \mathbf{y}$ given $\mathbf{X} = \mathbf{x}$, and by

Algorithm 8 Mixed SVRG Method

Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0$, and positive integers B and m.
for $k = 1, 2, \ldots$ **do**
  Sample a batch of size B without replacement from $\{1, ..., n\}$.
  Compute the batch gradient $\nabla R_B(w_k) = \sum_{i \in B} \nabla f_i(w_k)$.
  Initialize $\tilde{w}_1 \leftarrow w_k$.
  for $j = 1, \ldots, m$ **do**
    Chose i_j uniformly from $\{1, \ldots, n\}$.
    if $i_j \in B$ **then**
      Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_B(w_k))$.
    else
      Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j)$.
    Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_B(w_k))$.
    Set $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$.
  Choose j uniformly from $\{1, \ldots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.

$P(Y_i = b|\mathbf{x})$ the corresponding marginal distribution of Y_i :

$$P(Y_i = b|\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}: y_i = b} P(\mathbf{y}|\mathbf{x}).$$

In general, a multi-label classifier h is an $\mathcal{X} \to \mathbb{R}^m$ mapping that for a given instance $x \in \mathcal{X}$ returns a vector

$$\mathbf{h}(\mathbf{x}) = (h_1(x), h_2(x), ..., h_m(x)).$$

The precise problem of MLC can then be stated as follows: Given training data in the form of a finite set of observations $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$, drawn independently from $P(\mathbf{X}, \mathbf{Y})$, the goal is to learn a classifier $\mathbf{h} : \mathcal{X} \to \mathbb{R}^m$ that generalizes well beyond these observations in the sense of minimizing the risk with respect to a specific loss function. The risk of a classifier $\mathbf{h}$ is defined formally as the expected loss over the joint distribution $P(\mathbf{X}, \mathbf{Y})$:

$$R_L(\mathbf{h}) = \mathbb{E}_{\mathbf{XY}} L(\mathbf{Y}, \mathbf{h}(\mathbf{X})) \tag{12}$$

where $L(\cdot)$ is a loss function on multi-label predictions. The so-called risk-minimizing model $\mathbf{h}^*$ is given by

$$\mathbf{h}^*(\mathbf{x}) = \arg\min_{\mathbf{y}} \mathbb{E}_{\mathbf{Y}|\mathbf{X}} L(\mathbf{Y}, \mathbf{y}). \tag{13}$$

VI. LOSS FUNCTIONS

In this section, we describe the three loss functions being used for our particular analysis viz. - Hamming Loss, Subset 0/1 Loss and Gebru Loss. We also provide a theoretical analysis on Hamming and Subset Loss.

A trivial approach to solve MLC problems can be through the decomposition the labels into several binary classification problems. However, this approach has been criticized for ignoring information about the interdependencies between the labels which can be crucial in some cases. Since we aim to predict all the labels simultaneously, it is important to exploit any such dependencies. In our setup, Hamming loss does not consider the dependence amongst the labels,

whereas Subset and Gebru losses do take these dependencies into account.

*1) **Hamming Loss**:* is a simple loss function defined as the fraction of labels whose relevance is incorrectly predicted. It is defined as-

$$L_H(y, h(x)) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}[y_i \neq h_i(x)] \tag{14}$$

For the Hamming Loss (14), the risk minimizer (13) is obtained by

$$\mathbf{h}^*_H(\mathbf{x}) = (h_{H_1}(\mathbf{x}), ..., h_{H_m}(\mathbf{x})),$$

where

$$h_{H_i}(\mathbf{x}) = \arg\max_{b\in\{0,1\}} P(Y_i = b|\mathbf{x}) \quad (i = 1, ..., m). \tag{15}$$

However, since (14) is non-convex and non-differentiable, we choose logistic loss as a surrogate loss function-

$$f(\mathbf{w}; x, y) = \frac{1}{M}\sum_{i=1}^{M} \Big\{ \ln(1 + \exp(-y w_i^T x)) - \lambda_i ||w_i||^2 \Big\} \tag{16}$$

*2) **Subset Loss 0/1**:* is closely related to the estimation of a joint probability distribution. It can be described as-

$$L_S(y, h(x)) = \mathbb{1}[y \neq h(x)] \tag{17}$$

As shown in [8], the risk-minimizing prediction for (17) is simply given by the mode of the distribution:

$$\mathbf{h}^*_s(\mathbf{x}) = \arg\max_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x}) \tag{18}$$

This shows that the entire distribution of $\mathbf{Y}$ given $\mathbf{X}$, or at least enough knowledge to identify the mode of this distribution, is needed to minimize the Subset Loss. The derivation of a risk-minimizing prediction requires the modeling of the joint distribution (at least to some extent), and hence the modeling of conditional dependence between labels.

In order to deal with Subset Loss, we make two key decisions. We treat MLC as multi-class classification problem, where we treat each combination of labels to be one class. Here the set of classes become the label power-set. However, this leads to the possibility of 2^m number of classes, where m is number of labels. We use a cross-entropy loss functions as the surrogate for Subset Loss. Cross-entropy loss is a smooth and convex loss function, which is defined as follows:

$$f(\mathbf{w}; x, y) = -\sum_{i=1}^{m} y_i \log\Big(\frac{e^{w_i^T x}}{\sum_{j=1}^{m} e^{w_j^T x}}\Big) \tag{19}$$

Here, $y_k \in \{0, 1\}$ for $k \in \{1, 2, ..., m\}$ and w_k^T represents the row of the weight matrix $\mathbf{w}$ for label $k \in \{1, 2, ..., m\}$.

*3) **Gebru Loss**:* is similar in abstraction to Subset Loss in the sense that it is also related to the estimation of a joint probability distribution. While Subset Loss suffers from the disadvantage of a large number of possible classes (2^m); Gebru Loss only considers a probability distribution over the labels. Another distinction between the two is that Gebru Loss uses KL divergence as the metric to quantify this distance between the actual and the predicted probability distribution.

This method computes a softmax distribution across label categories for label a, $\hat{p}_a = [\hat{p}_{a_1}, ..., \hat{p}_{a_K}]$ using the computed label scores (the true distribution is defined as p_a). The Gebru Loss for each label a is then given as the symmetric version of the KL divergence between $\hat{p}_a$ and p_a -

$$f(\mathbf{w}; x, p_a) = \frac{1}{2} D_{KL}(p_a||\hat{p}_a) + \frac{1}{2} D_{KL}(\hat{p}_a||p_a) \tag{20}$$

We take our estimate of the distribution to be the softmax distribution over the labels. Hence, we have

$$D_{KL}(p_a||\hat{p}_a) = \sum_{i=1}^{m} p_{a_i} \log\Big(\frac{p_{a_i}\sum_{j=1}^{m} e^{w_j^T x}}{e^{w_i^T x}}\Big) \tag{21}$$

$$D_{KL}(\hat{p}_a||p_a) = \sum_{i=1}^{K} \frac{e^{w_i^T x}}{\sum_{j=1}^{m} e^{w_j^T x}} \log\Big(\frac{e^{w_i^T x}}{p_{a_i}\sum_{j=1}^{m} e^{w_j^T x}}\Big) \tag{22}$$

Here, $p_a \in \{0, 1/c\}$, c is the number of active labels and w_i^T represents the row of the weight matrix $\mathbf{w}$ for label $i \in \{1, 2, ..., m\}$.

A. *Theoretical Insights into MLC*[3]

In terms of loss minimization, there are two views prevalent in the literature. These are:

1) The individual label view (Hamming Loss): seeks to improve the prediction accuracy of a single label by utilizing information from other labels.
2) The joint label view (Subset and Gebru Loss): identifies and minimizes the proper non-decomposable MLC loss functions suitable for evaluating multi-label prediction as a whole.

A classifier supposed to be good for solving one of those problems may perform poorly for another problem. We discuss two losses - Hamming Loss and Subset Loss. For the analysis, we take unconstrained hypothesis space, which allows us to consider the conditional distribution for a given $\mathbf{x}$.

In this section, we show that, in general, the Hamming Loss minimizer and the Subset Loss minimizer will differ significantly. That is, the Hamming Loss minimizer may be poor in terms of the Subset 0/1 Loss and vice versa. Therefore, it becomes important to choose our algorithm in compliance with the loss functions.

Theorem 1: The Hamming Loss and Subset 0/1 have the same risk minimizer, i.e., $\mathbf{h}^*_H(\mathbf{x}) = \mathbf{h}^*_s(\mathbf{x})$, if one of the following conditions holds:

1) Labels $Y_1, ..., Y_m$ are conditionally independent, i.e., $P(\mathbf{Y}|\mathbf{x}) = \prod_{i=1}^{m} P(Y_i|x)$.
2) The probability of the mode of the joint probability is greater than or equal to 0.5, i.e., $P(\mathbf{h}^*_s(\mathbf{x})|\mathbf{x}) \geq 0.5$.

[3]The theoretical analysis is borrowed from [8], but written succinctly depending on the scope of this project. The proofs can be found in the same source.

Furthermore, the two loss functions are related to each other because of the following bounds.

Theorem 2: For all distributions of $\mathbf{Y}$ given $\mathbf{x}$, and for all models $\mathbf{h}$, the expectation of the Subset 0/1 loss can be bounded in terms of the expectation of the Hamming loss as follows:

$$\frac{1}{m}\mathbb{E}_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}(\mathbf{x}))] \leq E_{\mathbf{Y}}[L_H(\mathbf{Y}, \mathbf{h}(\mathbf{x}))] \leq E_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}(\mathbf{x}))]. \quad (23)$$

However, the next set of results show that using a classifier tailored for the wrong loss function may yield bad performance for the other loss. We define the regret of a classifier $\mathbf{h}$ with respect to a loss function L_z as follows:

$$r_{L_z}(\mathbf{h}) = R_{L_z}(\mathbf{h}) - R_{L_z}(\mathbf{h}^*_z) \quad (24)$$

where R is the risk given by (12), and $\mathbf{h}^*_z$ is the Bayes-optimal classifier with respect to the loss function L_z. The regret with respect to the Hamming Loss is given by

$$r_H(\mathbf{h}) = \mathbb{E}_{\mathbf{XY}}[L_H(\mathbf{Y}, \mathbf{h}(\mathbf{X}))] - \mathbb{E}_{\mathbf{XY}}[L_H(\mathbf{Y}, \mathbf{h}^*_H(\mathbf{X}))], \quad (25)$$

and that for Subset 0/1 Loss is given by

$$r_s(\mathbf{h}) = \mathbb{E}_{\mathbf{XY}}[L_s(\mathbf{Y}, \mathbf{h}(\mathbf{X}))] - \mathbb{E}_{\mathbf{XY}}[L_s(\mathbf{Y}, \mathbf{h}^*_s(\mathbf{X}))]. \quad (26)$$

Since both the loss functions are decomposable with respect to individual instances, we analyze the expectation over $\mathbf{Y}$ for a given $\mathbf{x}$. The first result concerns the highest value of the regret in terms of the Subset Loss for $\mathbf{h}^*_H(\mathbf{X})$, the optimal strategy for the Hamming Loss.

Theorem 3: The following upper bound holds:

$$\mathbb{E}_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}^*_H(\mathbf{x})] - \mathbb{E}_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}^*_s(\mathbf{x})] < 0.5. \quad (27)$$

Moreover, this bound is tight, i.e.,

$$\sup_P \mathbb{E}_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}^*_H(\mathbf{x})] - \mathbb{E}_{\mathbf{Y}}[L_s(\mathbf{Y}, \mathbf{h}^*_s(\mathbf{x})] < 0.5, \quad (28)$$

where the supremum is taken over all probability distributions on $\mathcal{Y}$.

The second result concerns the highest value of the regret in terms of Hamming Loss for $\mathbf{h}^*_s(\mathbf{X})$, the optimal strategy for Subset Loss.

Theorem 4: The following upper bound holds for $m > 3$:

$$\mathbb{E}_{\mathbf{Y}}[L_H(\mathbf{Y}, \mathbf{h}^*_s(\mathbf{x})] - \mathbb{E}_{\mathbf{Y}}[L_H(\mathbf{Y}, \mathbf{h}^*_H(\mathbf{x})] < \frac{m-2}{m+3}. \quad (29)$$

Moreover, this bound is tight, i.e.,

$$\sup_P \mathbb{E}_{\mathbf{Y}}[L_H(\mathbf{Y}, \mathbf{h}^*_s(\mathbf{x})] - \mathbb{E}_{\mathbf{Y}}[L_H(\mathbf{Y}, \mathbf{h}^*_H(\mathbf{x})] = \frac{m-2}{m+3}, \quad (30)$$

where the supremum is taken over all probability distributions on $\mathcal{Y}$.

As we can see, the worst case regret is high for both loss functions, suggesting that a single classifier will not be able to perform equally well in terms of both the loss functions. A classifier specifically tailored for the Hamming (Subset 0/1) Loss can perform very badly if evaluated on Subset Loss and vice versa. Hence, it becomes necessary to understand the loss functions and compare classifiers *w.r.t* to the main objective – which is essentially specified by the loss functions used for training. Although the above analysis does not directly relate with our current project goals, we will see that different variants of SGD perform differently on these loss functions – sophisticated variants performing well on more complicated loss functions such as Subset Loss and Gebru Loss. Nevertheless, we mention the above analysis as an interesting and critical issue with the MLC literature [8], [14]–[16].

VII. EXPERIMENTAL SETUP

Given the understanding of algorithms corresponding to SGD variants and loss functions, next we discuss the experimental analysis.

A. *Dataset*

For our study, we have used three MLC data sets, available at KEEL.[4]

1) Music-Emotions - This is a well-known dataset for the automated detection of emotion in music tailored as an MLC task. Here, a piece of music may belong to more than one emotion classes. There are 593 instances of songs with 72 real valued features. Each instance is labeled with multiple labels from a set of 6 labels.
2) MediaMills - This data set contains data from a generic video indexing problem where each item can belong to one or more classes. The data comprises of 43907 samples and 120 features. Each sample may belong to one or more of the 101 different labels.
   Owing to the very large size of this dataset, and possible larger number of binary classifier (2^{101}), we work with *two smaller* sets of data -
   - MediaMills - 7 Labels - We select seven labels at random and only use the data points that belong to at least one of these seven label categories. The resulting data set yields a dataset of size 3706×120.
   - MediaMills - 26 Labels - Similar to the previous case, we select 26 random labels yielding a data set of dimension 15687×120.

B. *Experiment*

For the analysis, we evaluate the performance of SGD and it's variants (Section IV) - across the three datasets - over the three different loss functions (Section VI). Since we are only interested in the relative performance of these algorithms under similar experimental settings, we have not fine-tuned the parameters for these algorithms and have used the same diminishing step size rule throughout all experiments viz. $\alpha = \frac{1}{k}$. The Emotions and MediaMills were normalized using a Min-Max-Scaler (each feature has range 0-1) and standard scaler (each feature has zero mean and unit variance), respectively. We choose these scaling rules as

[4]http://sci2s.ugr.es/keel/dataset_smja.php?cod=922#sub1

Data Set	Loss\Algorithm	SGD	SVRG 1	SVRG 2	SVRG 3	SAGA	SAG	S2GD	VR Lite	Batching Svrg	Mixing SVRG
Emotions	Hamming	0.27s	0.23s	0.23s	0.23s	1.02s	0.30s	0.23s	0.20s	0.21s	0.24s
	Subset	1.46s	2.27s	2.23s	2.23s	18.25s	2.97s	2.25s	1.70s	2.06s	2.03s
	Gebru	2.48s	3.07s	3.06s	2.96s	6.34s	2.81s	3.11s	2.65s	2.92s	3.01s
MediaMills - 7 Labels	Hamming	32.74s	32.52s	31.88s	32.73s	1m 50s	43.98s	34.78s	1.64s	33.96s	35.82s
	Subset	37.93s	39.38s	40.25s	39.69s	11m 24s	2m 03s	40.36s	7.46s	38.69s	39.37s
	Gebru	55.25s	57.20s	55.09s	55.11s	6m 57s	1m 42s	56s	22.04s	54.60s	55.85s
MediaMills - 26 Labels	Hamming	14m 40s	14m 33s	14m 27s	14m 36s	41m 48s	16m 45s	14m 33s	0m 8s	14m 30s	14m 32s
	Subset	18m 07s	20m 48s	20m 52s	20m 59s	N/A	7h 23m 26s	20m40s	4m31s	19m 50s	20m 04s
	Gebru	17m 51s	18m 44s	18m 34s	18m 29s	N/A	1h 04m 14s	18m 36s	3m 34s	19m 01s	19m 19s

TABLE I: Summary of results. The entries corresponds to the run time of algorithms for three loss functions and three datasets. Color coding has the following meaning: (a) Yellow - Similar performance to SGD, (b) Green - Superior performance to SGD, and (c) Red - Inferior performance to SGD.

they yielded the lowest condition number for their respective data sets.

We run each case for 5-epochs each. In a preliminary experiment, we experimented with the Emotions data set for up to 30 epochs. Since, we did not observe any significant differences in the relative performance of the algorithms when run for more epochs, we have decided to proceed with 5 epochs only. The C.P.U. time required for the execution of each algorithm across all cases was also logged. We now present the results obtained via our experiment, and their corresponding discussions.

VIII. RESULTS

Owing to the large number of possible cases (3 data sets, 3 loss functions and 10 optimization algorithms), we present the results data set wise. However, we present a summary of the results in Table I[5]. The table presents the execution time for each algorithm across all cases. Additionally, the performance of each algorithm (at 5 epochs) w.r.t. SGD has been color-coded. Yellow denotes similar performance as SGD, Green denotes superior performance and Red denotes inferior performance w.r.t SGD. Cells marked N/A denote no convergence was obtained within 12 hours.

A. *Emotions*

*1) **Hamming Loss**:* Fig. 2 shows the results for the hamming loss setting for the emotions datasets. We observe that all the algorithms exhibit similar convergence to the solution. While some algorithms exhibit more oscillations than others, the overall performance is effectively the same.

We attribute this behavior to the relatively lower condition number (720) for this particular data set along with the relatively lower complexity of the Hamming Loss setting. The lower condition number implies easier convergence. Generally, SGD converges quickly in the beginning and oscillates near the optimal (so called confusion zone) [17]. In this case, due to the low condition number, the algorithm does not enter the confusion zone and thus performs favorably. Additionally, remember that in our implementation,

[5]We will reference to this table numerous times in the upcoming sections and recommend the reader view these results in color for an easier understanding.

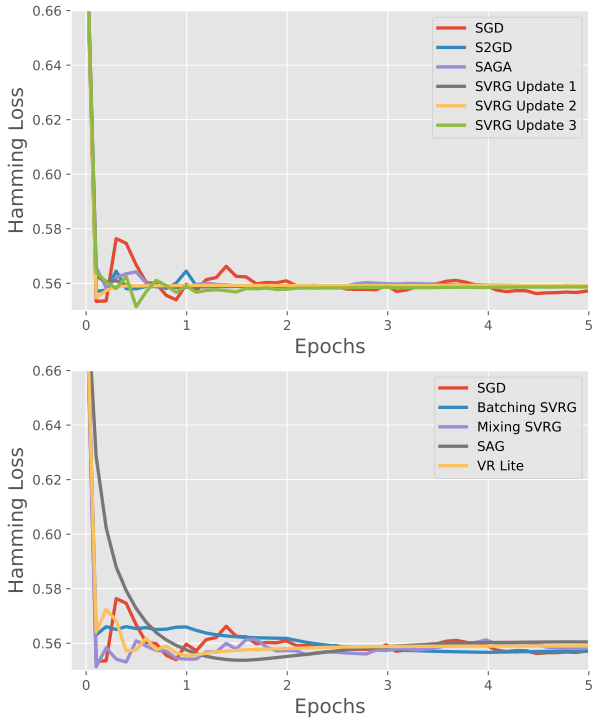

Fig. 2: Results for Emotions data set under the Hamming Loss setting. Performance of algorithms for this loss is similar; however, SGD shows more oscillations.

Hamming Loss is mathematically equivalent to logistic regression. Owing to the relative ease of this loss function, our baseline method (SGD) converges fast and so do the other algorithms. Thus, the lower condition number and the relative simplicity of Hamming Loss leads to similar performance for all algorithms.

In terms of execution times, all algorithms demonstrate similar behavior (around $0.25 - 0.3s$) except SAGA which takes about thrice the amount of time. The longer execution time of SAGA is consistent throughout all the cases. We attribute this behavior to the gradient storing property of SAGA. At each iteration, SAGA computes the gradient for each instance; stores the full gradient vector; and corrects the new gradient by using the mean of the gradients throughout the samples. We believe this storing and correction operation

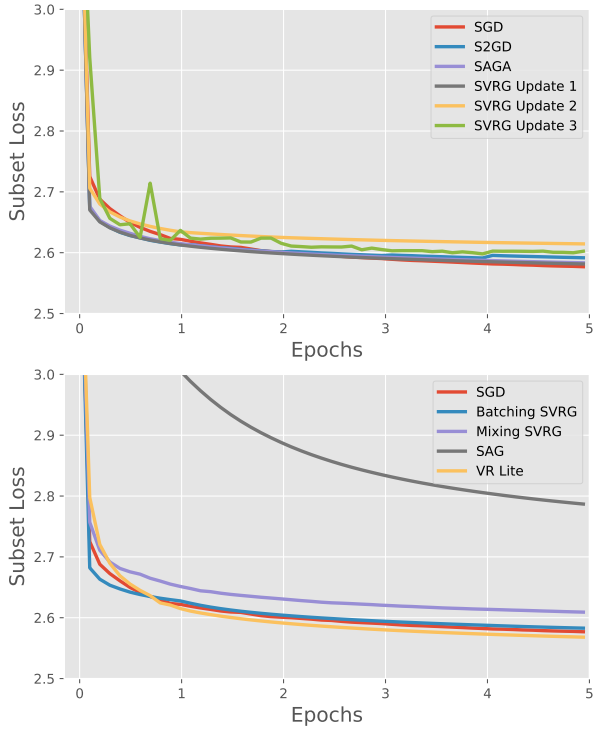

Fig. 3: Results for Emotions data set under the Subset Loss setting. SAG performs the worst for this loss.

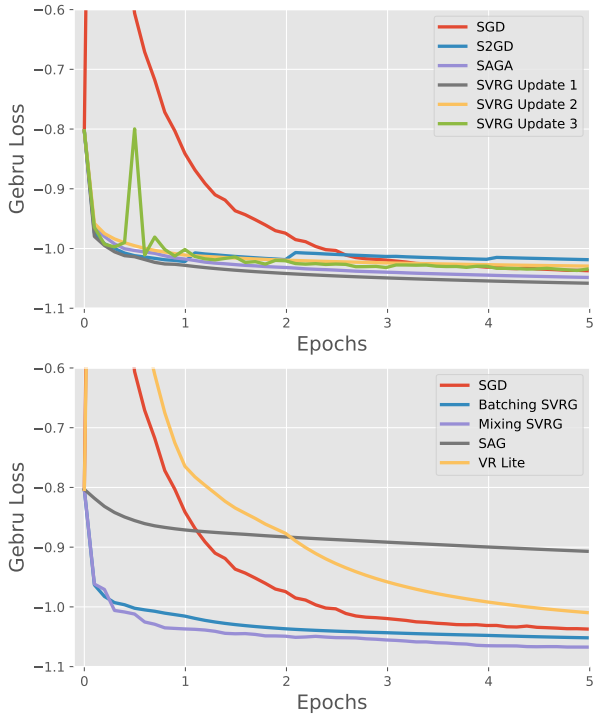

Fig. 4: Results for Emotions data set under the Gebru Loss setting. More sophisticated variants perform better.

lead to longer script execution times for SAGA.

*2) **Subset and Gebru Loss**:* In the case of Subset Loss (Fig.3) and Gebru Loss (Fig. 4), once again we observe that

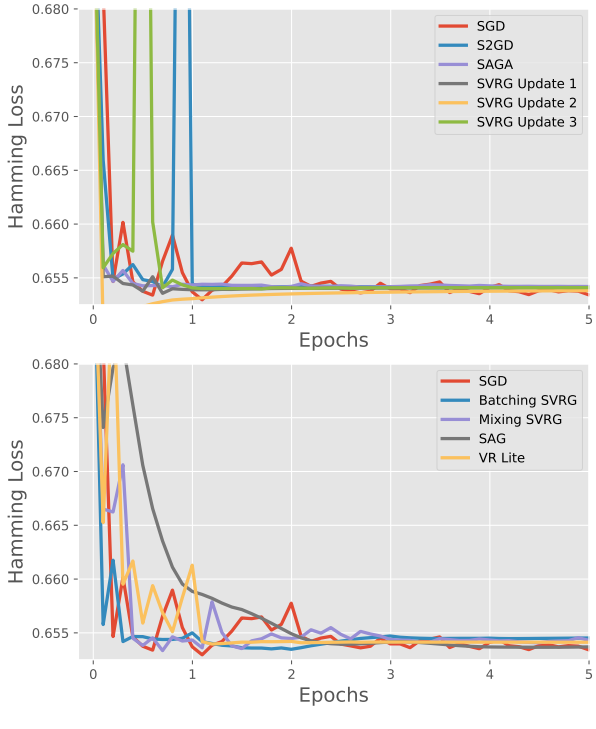

Fig. 5: Results for MediaMills - 7 Labels data set under the Hamming Loss setting. All the algorithms show drastic oscillations initially, but later become stable.

all algorithms exhibit a similar behavior. The only exception to this rule is SAG. Although SAG shows least oscillations, its performance to SGD is inferior for both the loss functions. Once again SAGA demonstrates longer execution times.

B. MediaMills - 7 Labels

*1) **Hamming Loss**:* As with the previous case, all algorithms demonstrate similar performance for the Hamming Loss setting (Fig 5). Both, SAGA and SAG demonstrate longer script execution times. Remember that SAGA and SAG use gradient memory to improve performance. This observation reaffirms out previous hypothesis that the memory storage and improvement step slows down the algorithm.

*2) **Subset Loss**:* In the case of this larger dataset, the superiority of the SGD variants becomes evident. In most cases, the SGD variants outperform SGD (Fig. 6). The only outlier is SVRG update 3, which shows a significantly inferior performance compared to SGD. As with the previous cases, SAGA and SAG exhibit longer script execution times.

*3) **Gebru Loss**:* Similar to the previous case most algorithms (with the exception of SVRG Update 3 and VR Lite) outperform SGD (Fig. 7).

It is interesting to note that as the loss functions become complicated and datasets become larger, more sophisticated algorithms start performing better. In the case of Hamming Loss, the performances of these algorithms are similar; whereas, in the case of Subset 0/1 and Gebru, the performances start to vary.

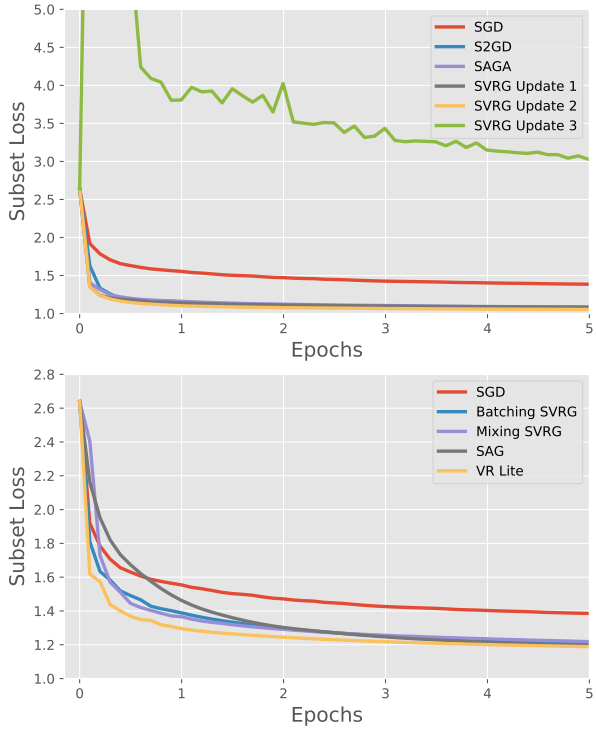

Fig. 6: Results for MediaMills - 7 Labels data set under the Subset Loss setting. More sophisticated algorithms perform better except SVRG (update 3).

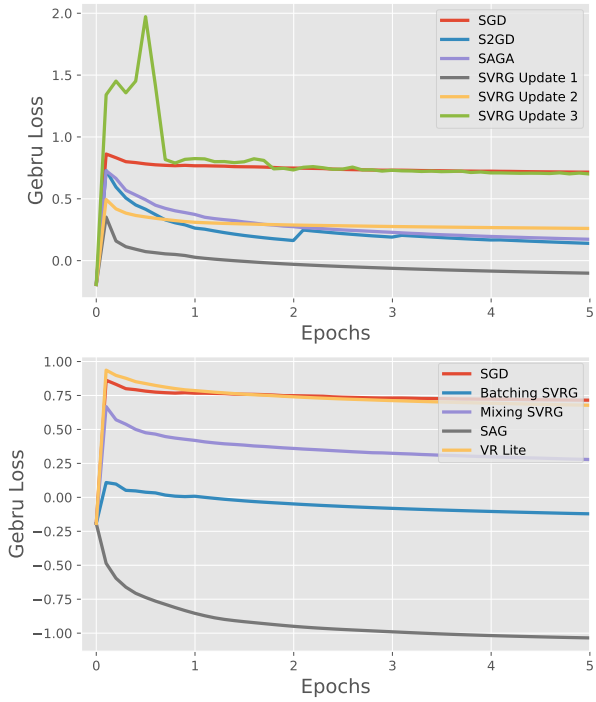

Fig. 7: Results for MediaMills - 7 Labels data set under the Gebru Loss setting. Huge variance in terms of performance. However, again, more sophisticated algorithms perform better except SVRG (update 3).

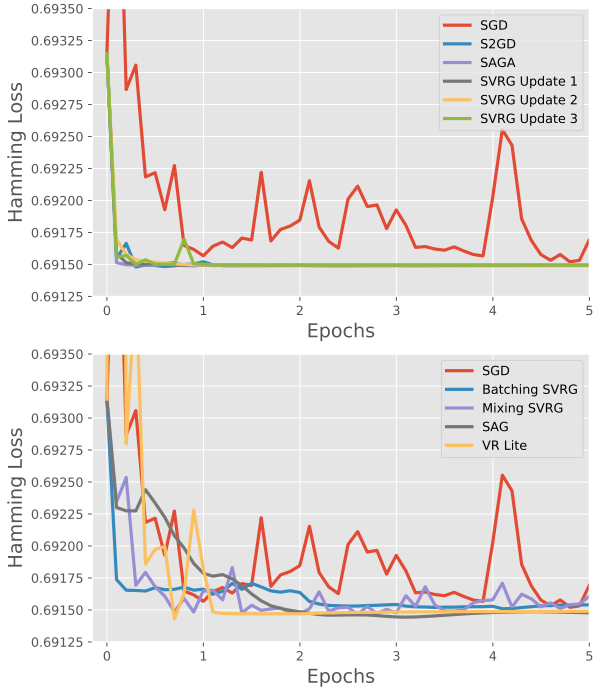

Fig. 8: Results for MediaMills - 26 Labels data set under the Hamming Loss setting. Here, we see difference between performance of SGD and other variants in the Hamming loss setting. SGD has oscillations right till the end, whereas others do not.

C. *MediaMills - 26 Labels*

In this case, we use the largest dataset with around $15K$ samples. Most methods outperform SGD with a few exceptions. We also observed that in some cases, the algorithms failed to converge within 12 hours after which we stopped the script execution.

*1) **Hamming Loss**:* All methods outperform SGD (Fig. 8). We observed that the condition number is very large in this case (around $32K$). We believe that in this case, despite the simplicity of the Hamming Loss function, SGD is stuck in the aforementioned confusion zone. This is evident due to the presence of large oscillations in the case of SGD. As opposed to previous cases, these oscillations do not damp out as the iterations progress (despite the use of a diminishing step size).

*2) **Subset Loss**:* Once again, most methods outperform SGD in the case of Subset Loss (Fig. 9). However, SAG shows slightly inferior performance to SGD, and takes around 7 hours to converge. In case of SAGA, the algorithm fails to converge in 12 hours.

*3) **Gebru Loss**:* While few algorithms outperform SGD, SVRG Update 2, VR-Lite and Batching SVRG show inferior performance when compared to SGD. Once again SAGA fails to converge for this particular case.

IX. DISCUSSIONS AND CONCLUSIONS

In this section, we summarize the results observed in the previous section and provide our reasoning for certain

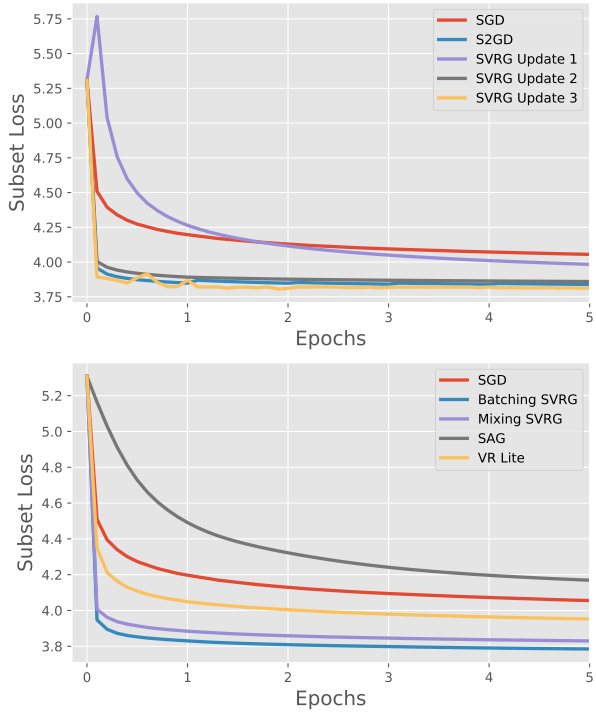

Fig. 9: Results for MediaMills - 26 Labels data set under the Subset Loss setting. All the other variants show no oscillations and perform better than SGD except SAG.

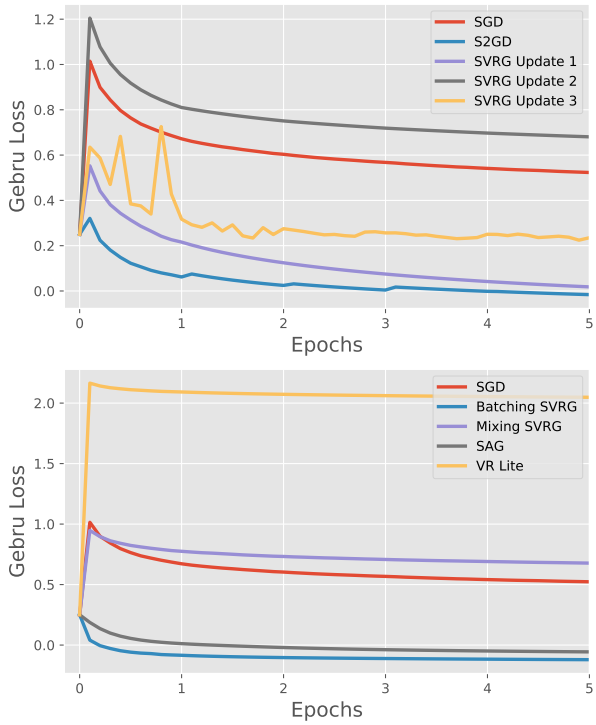

Fig. 10: Results for MediaMills - 26 Labels data set under the Gebru Loss setting. Again, except SAG, every variant performs better than SGD. Some show oscillations in the beginning but later become stable.

behaviors. Using these observations, we provide a guideline for readers on which algorithm/loss to use for which case (see Table II). We recommend that the user take these observations with a grain of salt as these observation may not hold true in all cases.

A. *Performance on Simple Case*

We observe that for simple cases with smaller datasets (few hundreds of samples) and smaller condition numbers, most algorithms perform similar to SGD for all three loss functions. A lower condition number implies easier convergence. Generally, SGD converges quickly in the beginning and oscillates near the optimal (so called confusion zone) [17]. In this case, due to the low condition number, the algorithm does not enter the confusion zone and thus performs favorably.

In these simple cases, Subset Loss demonstrates the most stable convergence for most cases with minimal oscillations.

B. *Performance on Medium Sized Data Sets*

In these cases (few thousand samples), the advantage of the SGD variants is highlighted. Most algorithms outperform SGD for the Subset and Gebru Loss conditions. Once again, Subset Loss demonstrates the most stable convergence behavior. However, all variants exhibit similar performance to SGD in the Hamming Loss setting.

We attribute this behavior to the relative simplicity of the Hamming Loss setting. Remember that in our implementation, the surrogates used for each label makes it mathematically equivalent to label-wise logistic regression. Owing to the relative ease of this loss function, our baseline method (SGD) converges fast and so do the other algorithms.

C. *Performance on Large Data Sets*

For larger data sets ($> 10K$ observations), once again Subset Loss exhibits the most stable convergence. For the Hamming Loss case, all algorithms outperform SGD. While most algorithms outperform SGD in the case of Subset and Gebru Loss, some algorithms demonstrate inferior performance and SAGA fails to converge for both cases.

D. *Execution time of SAG and SAGA*

SAG and SAGA use gradient memory to improve the quality of the weight estimates. However, they calculate the gradient at each iteration for all the instances, which leads to longer execution times for these algorithms. In one case we observed that SAG took 20 times longer than the average time for the other algorithms. In certain cases, SAGA failed to converge at all. Overall, we observed that the improvement in performance (if any) for these algorithms does not justify the longer script execution times for our particular case.

E. *Comment about Loss Functions*

We observe that the performance of the algorithms depends a lot on the complexity of the loss functions. Further, loss functions also play a role in defining the overall objective of the problem. We see that solution to the Hamming Loss can be (unboundedly) bad for the Subset Loss and vice versa.

TABLE II: Guidelines for use of SGD Variants and Loss Functions. !X signifies avoid using X.

Data Set Complexity	Low	Medium	High
Algorithm/Loss	SGD/Subset	SGD Variant/Subset	SGD Variant (!SAG, !SAGA)/Subset

Thus, it is important to use the same loss function to evaluate performance of the different algorithms. A trade-off needs to be made between simplicity of a loss function to optimize, and it being a true reflector of the main objective. We would like to extend the same analysis for the Gebru Loss in the future.

F. Comment about Gebru Loss

In this paper, we explored a relatively newer loss viz. the Gebru Loss [9]. While we were unable to conduct a detailed theoretical analysis on the said loss function, we have made a few critical observations for the same. Gebru Loss uses a symmetric KL divergence between the predicted and the actual label distribution as a measure of performance. In our experiments, we observed that Gebru Loss is highly sensitive to the step size choice and failed to converge for numerous cases. One reason might be the division by zero in (22). Conversely, the term inside the logarithmic in (21) becomes zero in some cases, leading to numerical issues. Similar problems were observed in the gradient of the loss function. In order to deal with these cases, we introduced a set of conditions in our code, thus compromising on the robustness of this loss function. While this loss function may be effective for specific cases, it can not be generalized for a multitude of problem sets easily.

REFERENCES

[1] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[2] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 315–323. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999611.2999647

[3] N. L. Roux, M. Schmidt, and F. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 2663–2671. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999325.2999432

[4] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1646–1654.

[5] J. Konen and P. Richtrik, "Semi-stochastic gradient descent methods," *Frontiers in Applied Mathematics and Statistics*, vol. 3, p. 9, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fams.2017.00009

[6] S. De, G. Taylor, and T. Goldstein, "Variance Reduction for Distributed Stochastic Gradient Descent," *ArXiv e-prints*, Dec. 2015.

[7] R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen, "Stopwasting my gradients: Practical svrg," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2251–2259. [Online]. Available: http://papers.nips.cc/paper/5711-stopwasting-my-gradients-practical-svrg.pdf

[8] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence and loss minimization in multi-label classification," *Machine Learning*, vol. 88, no. 1-2, pp. 5–45, 2012.

[9] T. Gebru, J. Hoffman, and L. Fei-Fei, "Fine-grained recognition in the wild: A multi-task domain adaptation approach," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 1358–1367.

[10] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas, "Multi-label classification of music into emotions." in *ISMIR*, vol. 8, 2008, pp. 325–330.

[11] "Mulan a java library for multi-label learning," http://mulan.sourceforge.net/datasets-mlc.html, accessed: 2018-04-25.

[12] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[13] A. Agarwal, P. L. Bartlett, P. Ravikumar, and M. J. Wainwright, "Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization," *Information Theory, IEEE Transactions on*, vol. 99, pp. 1–1, 2010.

[14] K. Dembczynski, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains." in *ICML*, vol. 10, 2010, pp. 279–286.

[15] K. Dembszynski, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence in multilabel classification," in *LastCFP: ICML Workshop on Learning from Multi-label data*. Ghent University, KERMIT, Department of Applied Mathematics, Biometrics and Process Control, 2010.

[16] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier, "Regret analysis for performance metrics in multi-label classification: the case of hamming and subset zero-one loss," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 280–295.

[17] "Lecture Notes ie510 non linear programming - spring 2018," Dr Ruoyu Sun, accessed: 2018-05-12.