Small Batch or Large Batch? Gaussian Walk with Rebound Can Teach

Peifeng Yin
IBM Almaden Research Center
San Jose, CA, USA
peifengy@us.ibm.com

Ping Luo
Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS
Beijing, China
University of Chinese Academy of Sciences
Beijing, China
luop@ict.ac.cn

Taiga Nakamura
IBM Almaden Research Center
San Jose, CA, USA
taiga@us.ibm.com

ABSTRACT

Efficiency of large-scale learning is a hot topic in both academic and industry. The *stochastic gradient descent (SGD)* algorithm, and its extension *mini-batch SGD*, allow the model to be updated without scanning the whole data set. However, the use of approximate gradient leads to the uncertainty issue, slowing down the decreasing of objective function. Furthermore, such uncertainty may result in a high frequency of meaningless update on the model, causing a communication issue in parallel learning environment. In this work, we develop a *batch-adaptive stochastic gradient descent (BA-SGD)* algorithm, which can dynamically choose a proper batch size as learning proceeds. Particularly on the basis of Taylor extension and central limit theorem, it models the decrease of objective value as a Gaussian random walk game with rebound. In this game, a heuristic strategy of determining batch size is adopted to maximize the utility of each incremental sampling. By evaluation on multiple real data sets, we demonstrate that by smartly choosing the batch size, the BA-SGD not only conserves the fast convergence of SGD algorithm but also avoids too frequent model updates.

1 INTRODUCTION

Efficiency of large-scale learning is a hot topic in both academic and industry. Usually for differentiable objective function, gradient descent algorithm is applied to train a model. However, for a large-scale data, there is a high time cost in scanning the whole dataset to calculate the gradient in one iteration [4]. To solve this problem, *stochastic gradient descent (SGD)* and its extension *mini-batch SGD* become alternative learning algorithms and have been widely used.

The core idea of SGD is to approximate the gradient by a single data instance. To reduce the uncertainty of approximation, mini-batch SGD uses a batch of instances, where the batch size is a predetermined variable and is kept constant during learning. Many works focus on accelerating the SGD via adjusting learning rate [10, 25, 38], modifying approximate gradient [26, 28], averaging learned parameters [27] and parallelism [14, 23, 29, 31, 39].

In this work, we focus on how to dynamically determine the batch size during learning process. The approximate gradient can be treated a random variable with some degree of uncertainty. Intuitively when the uncertainty degree is low, a small batch size is preferred. On the other hand, if the uncertainty degree is high, a larger batch size should be used. The challenge is that there lacks a way of quantifying such uncertainty. Neither is there a strategy to determine a proper batch size with regarding to different uncertainty degrees.

To address this problem, we propose models for both approximate derivative and the learning process. With the support of Taylor expansion and Central Limit Theorem, we quantify the noise (uncertainty degree) with a Gaussian distribution. Furthermore, the learning process is then modeled as a random walk game with a Gaussian dice. Under such game setting, we discuss the limitation of mini-batch SGD in learning and describe a heuristic strategy of allocating batch under different situations, aiming at maximizing the efficiency of decreasing objective function value. Combining this strategy with SGD learning algorithm, we develop the *Batch-adaptive SGD (BA-SGD)* algorithm. Via evaluation on multiple real data sets, we demonstrate that our BA-SGD algorithm combines the advantage of mini-batch SGD and gradient descent learning algorithm and achieves the best performance.

In summary, the contributions of this work are as below:

- We model the individual derivative as a random variable that consists of a constant (real global derivative) plus a zero-mean noise term. By applying the central limit theorem, the sample gradient can thus be modeled as a normal distribution whose mean is the real gradient and the variance is the noise variance over batch size.
- We model the learning process as a random walk game with Gaussian dice, which helps reveal the limitation of mini-batch SGD during learning.
- We propose a Batch-adaptive SGD algorithm that adopts a heuristic strategy of determining batch in each iteration, aiming to maximize the efficiency of decreasing objective value with scanned training instances.

KDD'17, August 13–17, 2017, Halifax, NS, Canada.
 ISBN 978-1-4503-4887-4/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3097983.3098147

- We conduct experiments on multiple real data sets, demonstrating our method's better performance with regarding to i)fast convergence rate and ii) less frequent model update.

The rest of the paper is organized as follows. Section 2 gives a literature review of related work. Section 3 to 4 describes the models of stochastic derivative and learning process. Section 5 describes details of our proposed BA-SGD algorithm. Section 6 displays the experimental results on both real and synthetic data. Section 7 concludes this work and provides future work.

2 RELATED WORK

Efficiency of learning algorithm is an important task in machine learning, especially in scenario of large-scale data. For optimizing algorithms based on gradient, there are generally four directions, i.e., parallelism, stochastic, learning rate and variance reduction, of which all are sometimes combined with each other.

In large-scale data learning, it is found that the bottleneck lies in the scan of the whole data set to compute the gradient [4]. A straightforward way is thus to parallelize this part. Due to the parallel framework MapReduce [8] and the corresponding open source version Hadoop [11], many machine learning libraries, such as Mahout [12], Spark [37], MLI [33], etc. are developed. Also there are independent works such as GraphLab [22], DistBelief [7], TensorFlow [1] and other parameter-server-based designs [2, 16, 19]. Instead of parallel framework, another method, namely *alternating direction method of multipliers (ADMM)* [13, 15], is to transform the original optimization problem into one that is easily parallelized. Examples of its application in machine learning can be found in [5]. In [6], Chen et. al. extends the original two-block form to multiple blocks.

To avoid scanning the whole data set to compute the gradient, *stochastic gradient descent (SGD)* and its variant *mini-batch SGD* approximate this value by randomly selecting a single or multiple data example(s) from the whole data set. Particularly the batch size of mini-batch SGD is a predetermined constant. To accelerate SGD's speed, momentum [28] and Nesterov [26] are applied to modify the gradient. Works [14, 23, 29, 31, 39] studied the SGD implementation in parallel environment. In [20], Li et. al. proposed an update mini-batch strategy which adds a conservative penalty when updating parameters for each round. The aim is to retain small communication cost with large batch size but at the same maintains a good convergence rate in parallel environment.

There are a bunch of works studying the learning rate. Work [4] mentioned second-order gradient descent, which sets the learning rate to be the inverse of the objective function's Hessian matrix. It is claimed to achieves quadratic convergence [9]. Aslo in [4], the author proposed to make the learning rate of SGD decrease as the reverse of iteration number, as [25] shows this is the decreasing speed of error. Duchi [10] et. al. proposed *Adagrad* algorithm which decreases the learning rate based on the sum of all previous gradients. Other extensions include *AdaDelta* [38] and Adam [18]. The former one introduced a time-decay variable for Adagrad while the latter one combines momentum and AdaDelta.

Finally, a line of works focus on reducing SGD's variance caused by adoption of sample gradient. One major effort is to adjust/smooth the calculated stochastic gradient with historical ones, e.g., stochastic average gradient (SAG) [30], stochastic dual coordinate ascent (SDCA) [32], stochastic variance reduced gradient (SVRG) [17] and proximal stochastic gradient [35]. Another trial is to adopt non-uniform sampling strategies when estimating gradient, as proposed in [34].

Different from these previous works, we focus on studying how to automatically choose a good batch size during learning process. By modeling the stochastic derivative and the learning process, we propose a heuristic strategy of allocating batch against different situations of uncertainty. Thus our work can be easily combined with existing works about parallelism and learning rate.

3 MODEL OF STOCHASTIC DERIVATIVE

In this section we starts with a brief review of gradient descent learning algorithm and then propose our model of stochastic derivative used in SGD. Before moving forward, we summarize all symbols used in this work in Table 1 for easy reference.

3.1 Preliminary

The process of model training is an optimizing (minimization) problem. Mathematically, given a labeled data set $\mathcal{X}$ and a defined objective function $\mathsf{F} : R^d \to R$, the goal is to find a proper parameter configuration $\vec{\theta}$ so that the objective value is minimized, i.e., $\vec{\theta}^* = \arg\min_{\vec{\theta}} \mathsf{F}(\vec{\theta}|\mathcal{X})$.

It is usually assumed that each individual instance of the data set is independent of each other. Thus, the objective function can be written as the average of sub-objective ones, i.e., $\mathsf{F}(\vec{\theta}|\mathcal{X}) = \frac{1}{|\mathcal{X}|}\sum_{\mathbf{x}_i \in \mathcal{X}} \mathsf{G}^i(\vec{\theta})$[1].

In most problems, the objective function is (or exactly, purposely defined to be) differentiable. And the gradient descent algorithm can be used to learn the optimal parameters. The algorithm initializes the model with random parameter values and iteratively updates the model with gradients. Formally, let $\mathsf{f}_{\vec{\theta}} : R^d \to R^d$ denote the gradient (global derivative) and the $\mathsf{g}^i_{\vec{\theta}} : R^d \to R^d$ denote the individual instance gradient (local derivative), the update can be represented as the equation below.

$$\vec{\theta}' = \vec{\theta} - \eta \cdot \mathsf{f}_{\vec{\theta}} = \vec{\theta} - \eta \cdot \frac{\sum_{\mathbf{x}_i \in \mathcal{X}} \mathsf{g}^i_{\vec{\theta}}}{|\mathcal{X}|} \tag{1}$$

Here the η is defined as *learning rate*, whose value initially ranges from 0.01 to 0.1 and will be decreased if the objective value is observed to increase after parameter update.

For large scale learning problem, scanning the whole data set to compute the differential is quite time-consuming. Alternatively, for each iteration a subset (also known as *batch*) $\{\mathcal{Y} \subset \mathcal{X} || \mathcal{Y}| << |\mathcal{X}|, \}$ is randomly sampled to calculate the approximate derivative, i.e., $\mathsf{f}_{\vec{\theta}} \approx \hat{\mathsf{f}}_{\vec{\theta}} = \frac{\sum_{y_j \in \mathcal{Y}} \mathsf{g}^j_{\vec{\theta}}}{|\mathcal{Y}|}$. This method is known as *Stochastic Gradient Descent (SGD)*. Although the learning efficiency is improved, one issue of SGD is hard to determine when to decrease the learning rate. Conventional metric of objective value change does not work

[1] To avoid over-fit, there is usually a data-independent regularization function. In this case, it is easy to transform to the focused form. For instance, $\mathsf{h}(\vec{\theta}) = \mathsf{h}(\vec{\theta}|\mathcal{X}) = \frac{1}{|\mathcal{X}|}\sum_{\mathbf{x}\in\mathcal{X}} \mathsf{h}(\vec{\theta}|\mathbf{x}) \cdot |\mathcal{X}|$

Table 1: List of Symbols

Symbol	Description
$\mathcal{X}, \mathcal{Y}$	Data set and its random subset
d	Dimension of model parameters
$\vec{\theta} : R^d$	Vector of model parameters
η	Learning rate or moving speed in Random Walk game
$\mathsf{F} : R^d \to R$	Objective function on whole data set
$\mathsf{f}_{\vec{\theta}} : R^d \to R^d$	Partial derivative of function F
$\mathsf{G}^i : R^d \to R$	Sub objective function on single data sample $\mathbf{x}_i \in \mathcal{X}$
$\mathsf{g}^i_{\vec{\theta}} : R^d \to R^d$	Partial derivative of function G^i
$\vec{\xi}_i \equiv_{\textbf{def}} \mathsf{g}^i_{\vec{\theta}} - \mathsf{f}_{\vec{\theta}}$	Derivative difference between an individual instance and the whole data set
$\vec{\varepsilon}_{\mathcal{Y}} \equiv_{\textbf{def}} \frac{\sum_{\mathbf{y}_j \in \mathcal{Y}} \vec{\xi}_j}{\lvert\mathcal{Y}\rvert}$	Average of sampled individual derivative difference
S^*, S, s_t	State in Random Walk game
μ_t, σ_t^2	The mean and variance of Gaussian dice in Random Walk game at t^{th} round
m	investment in Random Walk game
$\mathbf{E}^{s_t}_m(s_{t+1})$	Expected state of time $t+1$ given current state s_t and investment m

any more, because now the "objective value" is also approximate based on the sampled batch $\mathcal{Y}$ and the increase or decrease could be ascribed to the estimation inaccuracy of either gradient or objective value itself. A common solution is to start with an even smaller learning rate and keep it constant. Furthermore, there is work [4] suggesting decrease the learning rate as the speed of $O(T^{-1})$, where T is the total number of iterations so far.

3.2 Stochastic Derivative

Empirical studies reveal that learning with SGD leads to the fluctuation of objective value, especially in later training stage. We may use the concept of individual derivative difference to explain it. Formally, let $\vec{\xi}_i \in R^d$ denote the difference between an individual instance's $\mathsf{g}^i_{\vec{\theta}}$ and the real one $\mathsf{f}_{\vec{\theta}}$. The sum of them is equal to zero vector, i.e., $\sum_{\mathbf{x}_i \in \mathcal{X}} \vec{\xi}_i = \mathbf{0}$. In SGD, the approximate derivative can thus be written to the following equation.

$$\hat{\mathsf{f}}_{\vec{\theta}} = \frac{\sum_{\mathbf{y}_j \in \mathcal{Y}} \mathsf{g}^j_{\vec{\theta}}}{|\mathcal{Y}|} = \frac{\sum_{\mathbf{y}_j \in \mathcal{Y}} (\mathsf{f}_{\vec{\theta}} + \vec{\xi}_j)}{|\mathcal{Y}|} = \mathsf{f}_{\vec{\theta}} + \frac{\sum_{\mathbf{y}_j \in \mathcal{Y}} \vec{\xi}_j}{|\mathcal{Y}|} \tag{2}$$

The second term $\frac{\sum_{\mathbf{y}_j \in \mathcal{Y}} \vec{\xi}_j}{|\mathcal{Y}|}$ can be treated as a random variable depending on the randomly sampled batch $\mathcal{Y}$. As *Central Limit Theorem (CLT)* points out, it satisfies a multi-dimension normal distribution $\mathcal{N}(\mathbf{0}, \frac{\Sigma}{|\mathcal{Y}|})$, where Σ is the covariance matrix of $\vec{\xi}_i$.

As can be seen from Equation (2), the estimation of gradient is affected by the scale of the "noise" variance. Take a one-dimension case as an example. Suppose the variance is constantly at scale of 1. At early stage of learning, the real gradient may be at scale of 10. In this case, the estimation is quite close to real one and the objective value will decrease rather quickly. However, as learning moves forward and the real gradient is decreasing, the impact becomes larger. If the gradient is at the scale of 1 or smaller, the estimated one is thus dominated by the noise term and may be completely reversed to real one. This scenario explains the high fluctuation in learning with SGD.

Another point is that the variance is correlated with the inverse of batch size $|\mathcal{Y}|^{-1}$. Therefore by increasing the batch size we can mitigate the fluctuation issue. For mini-batch SGD, the batch size is arbitrarily determined and kept constant through the whole learning process. A wiser strategy would dynamically update this value as learning proceeds, which is an motivation for our solution of *batch-adaptive SGD*.

4 MODEL OF STOCHASTIC LEARNING PROCESS

In previous section we claimed that the approximate derivative from a random sample satisfies a Gaussian distribution. Here we move a step further to model the learning process, or exactly, the decrease of objective value for each iteration.

4.1 Decrease of Objective Value

Given an objective function $\mathsf{F}(\vec{\theta})$, we may write it in first-order Taylor polynomial at any parameter configuration, e.g., $\vec{\theta}_0$, as shown in the equation below.

$$\begin{aligned} \mathsf{F}(\vec{\theta}) &= \mathsf{F}(\vec{\theta}_0) + \sum_{k=1}^{d} \mathsf{f}_{\vec{\theta}_0^{(k)}} \cdot (\vec{\theta}^{(k)} - \vec{\theta}_0^{(k)}) + \mathsf{h}_{\vec{\theta}_0}(\vec{\theta}) \\ &= \mathsf{F}(\vec{\theta}_0) + \mathsf{f}^T_{\vec{\theta}_0} \cdot (\vec{\theta} - \vec{\theta}_0) + \mathsf{h}_{\vec{\theta}_0}(\vec{\theta}) \end{aligned} \tag{3}$$

where the $\vec{\theta}^{(k)}$ represents the k-th element in the d-dimension vector and $\mathsf{f}^T_{\vec{\theta}_0}$ stands for the transpose of the matrix.

Here the function $\mathsf{h}_{\vec{\theta}_0}(\vec{\theta})$ is the remainder term whose value is 0 if the variable is close to $\vec{\theta}_0$, i.e., $\lim_{\vec{\theta} \to \vec{\theta}_0} \mathsf{h}_{\vec{\theta}_0}(\vec{\theta}) = 0$.

In SGD, model parameter is updated with an estimated gradient $\hat{\mathsf{f}}_{\vec{\theta}_0}$. With the help of our error-term model in Section 3, or exactly,

Equation (2), we can express the objective value change as follows.

$$\begin{aligned} &\mathsf{F}(\vec{\theta}_0 - \eta\hat{\mathsf{f}}_{\vec{\theta}_0}) - \mathsf{F}(\vec{\theta}_0) \approx \mathsf{f}_{\theta_0}^T(\theta_0 - \eta\cdot\hat{\mathsf{f}}_{\theta_0} - \theta_0) = -\eta\mathsf{f}_{\vec{\theta}_0}^T\hat{\mathsf{f}}_{\vec{\theta}_0} \\ &= -\eta\mathsf{f}_{\vec{\theta}_0}^T\cdot(\mathsf{f}_{\vec{\theta}_0} + \frac{\sum_{\mathrm{y}_j\in\mathcal{Y}}\vec{\xi}_j}{|\mathcal{Y}|}) = -\eta\cdot\mathsf{f}_{\vec{\theta}_0}^T\mathsf{f}_{\vec{\theta}_0} - \eta\cdot\mathsf{f}_{\vec{\theta}_0}^T\cdot\vec{\varepsilon}_{\mathcal{Y}} \end{aligned} \tag{4}$$

The first term is a constant (for that iteration) and thus the change only depends on the random variable $\vec{\xi}_j$. With CLT we know it satisfies a Gaussian distribution $\vec{\varepsilon}_{\mathcal{Y}} \sim \mathcal{N}(\mathbf{0}, \frac{\Sigma}{|\mathcal{Y}|})$, and thus the weighted sum of this vector satisfies a one-dimension Gaussian distribution, i.e., $\vec{\varepsilon}_{\mathcal{Y}}^T\cdot\hat{\mathsf{f}}_{\vec{\theta}_0} \sim \mathcal{N}(0, \frac{\hat{\mathsf{f}}_{\vec{\theta}_0}^T\cdot\Sigma\cdot\hat{\mathsf{f}}_{\vec{\theta}_0}}{|\mathcal{Y}|})$.

Now the only unknown part is the global covariance matrix Σ, whose unbiased estimation $\hat{\Sigma}$ is the adjusted covariance of randomly sampled batch $|\mathcal{Y}|$.

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{|\mathcal{Y}|-1}\sum_{\mathrm{y}_j\in\mathcal{Y}}\left(\mathrm{g}_{\vec{\theta}_0}^j - \frac{\sum_{\mathrm{y}_j\in\mathcal{Y}}\mathrm{g}_{\vec{\theta}_0}^j}{|\mathcal{Y}|}\right)\left(\mathrm{g}_{\vec{\theta}_0}^j - \frac{\sum_{\mathrm{y}_j\in\mathcal{Y}}\mathrm{g}_{\vec{\theta}_0}^j}{|\mathcal{Y}|}\right)^T \\ &= \frac{\sum_{\mathrm{y}_j\in\mathcal{Y}}(\mathrm{g}_{\vec{\theta}_0}^j - \hat{\mathsf{f}}_{\vec{\theta}_0})(\mathrm{g}_{\vec{\theta}_0}^j - \hat{\mathsf{f}}_{\vec{\theta}_0})^T}{|\mathcal{Y}|-1} \end{aligned} \tag{5}$$

4.2 Game of Gaussian Walk with Rebound

Equation (4) shows that for SGD learning algorithm, the decrease of objective value satisfies a Gaussian distribution where the mean is determined by the global derivative and the variance is affected by the batch size. Here we abstract the process as a random walk game with a Gaussian dice.

Particularly, we define the domain of game state as a half closed set of real numbers $[S^*, +\infty)$. The game starts with a random state and the goal is to move as close as possible to S^*. For each round, the player pays money for a *Gaussian Dice* and moves according to the resulted Gaussian value.

The mean of the Gaussian dice is solely determined by the current state. The variance is jointly controlled by the state and the player's investment. Particularly at state S_i, the generated moving steps satisfies such Gaussian distribution $\triangle s_i \sim \mathcal{N}\left(\mu_i, \frac{\sigma_i^2}{m}\right)$, where the $m \in [1, +\infty)$ stands for the player's investment. The game thus ends when the player runs out of the budget.

Connecting this game back to SGD learning, the space of hidden states represents all possible objective values and the particular S^* is thus the minimum objective value that learning can best achieve. Each state's parameters are respectively corresponding to the square sum of the gradient $\mathsf{f}_\theta^T\cdot\mathsf{f}_\theta$ and its multiplication with covariance matrix of noise $\sqrt{\mathsf{f}_\theta^T\cdot\Sigma\cdot\mathsf{f}_\theta}$. Intuitively, the decay m and transfer speed η are the batch size and learning rate.

This game of random walk is a continuous Markov process with infinite states. Given two state $\langle S_i, S_j\rangle$, there are generally two paths of state transfer, corresponding to two situations during learning process. We illustrate this scenario in Figure 1. As can be seen, for convex optimizing, there are usually two ways of transfer from one state (objective value) to another. For a normal gradient descent process, the S_i is directly decreased to S_j. Alternatively, S_i first reaches the minimum point S^* and then rebounds to S_j.

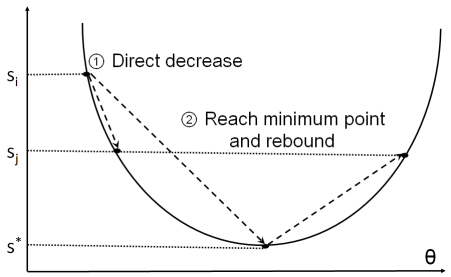


Figure 1: Illustration of State Transfer in Learning

To put it another way, for state $s_t = S_i$, let $\triangle s_t$ denote the steps generated by a Gaussian dice $\mathcal{N}(\mu_i, \frac{\sigma_i^2}{m})$. We define the next state s_{t+1} can be represented as in Equation (6).

$$s_{t+1} = |s_t - S^* - \eta\triangle s_t| + S^* = \begin{cases} s_t - \eta\triangle s_t, & \text{if } s_t - \eta\triangle s_t \geq S^* \\ 2S^* + \eta\triangle s_t - s_t, & \text{otherwise} \end{cases} \tag{6}$$

In this game setting, we can calculate the expected value of next state given a particular investment. Formally, let s_t denote the state at t^{th} round and μ_t, σ_t respectively stand for the associated Gaussian dice parameter. Given an investment of m, let $p_m^{s_t}(\triangle s_t)$ denote the probability density function for a random moving step $\triangle s_t \sim \mathcal{N}\left(\mu_t, \frac{\sigma_t^2}{m}\right)$. The expected value of next state can be expressed as below.

$$\begin{aligned} \mathbf{E}_m^{s_t}(s_{t+1}) &= \int_{-\infty}^{+\infty} p_m^{s_t}(\triangle s_t)(|s_t - S^* - \eta\cdot\triangle s_t| + S^*)d\triangle s_t \\ &= \eta\int_{-\infty}^{+\infty} p_m^{s_t}(\triangle s_t)\left|\triangle s_t - \frac{s_t - S^*}{\eta}\right|d\triangle s_t + S^* \\ &= (s_t - S^* - \eta\mu_t)\{\Phi(a) - \Phi(-a)\} + \frac{\eta\sigma_t}{\sqrt{m}}\sqrt{\frac{2}{\pi}}e^{-\frac{a^2}{2}} + S^* \\ &\text{where } a = \frac{s_t - S^* - \eta\mu_t}{\eta\sigma_t}\sqrt{m} \end{aligned} \tag{7}$$

This is a monotonically decreasing function for m. Consider the case when $m \to +\infty$, the $\Phi(a) - \Phi(-a)$ will converge to either 1 or -1, depending on the sign of a. The second term will be zero. Therefore, we have:

$$\lim_{m\to+\infty}\mathbf{E}_m^{s_t}(s_{t+1}) = \mathtt{sign}(a)(s_t - S^* - \eta\mu_t) + S^* = |s_t - S^* - \eta\mu_t| + S^* \tag{8}$$

To further illustrate Equation (7), we plot the correlation between state change $s_t - \mathbf{E}_m^{s_t}(s_{t+1})$ and the Gaussian variance σ_t^2 in Figure 2, where other parameters e.g., the batch size m, the μ, etc. are kept the same. As can be seen, in low-variance case, state is expected to become small (i.e., $s_t > \mathbf{E}_m^{s_t}(s_{t+1})$), but the decrease is bounded, confirmed with Equation (8). On the other hand, when variance is high, the state is expected to increase (i.e., $s_t < \mathbf{E}_m^{s_t}(s_{t+1})$. To avoid this case, one should increase the batch size m.

With Equation (7), we can evaluate different investment strategies. As illustrated in Figure 3, for a specific strategy that returns an investment m_t for a state s_t, if $\mathbf{E}_m^{s_t}(s_{t+1}) < s_t$, this strategy will make the player move forward in the game (i.e., state decrease). On the other hand, if $\mathbf{E}_m^{s_t}(s_{t+1}) > s_t$, the player will step backward

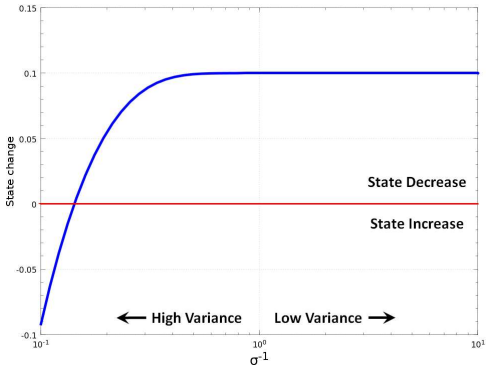


Figure 2: Correlation between State Change $s_t - \mathbf{E}_m^{s_t}(s_{t+1})$ and Variance σ_t^2. Note that the x-axis is σ^{-1}.

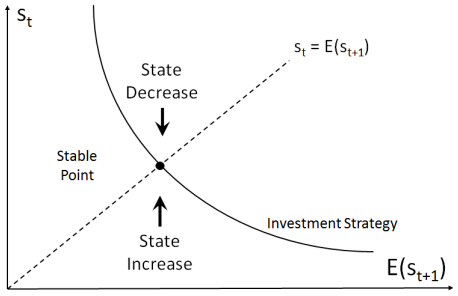


Figure 3: Three Scenarios of an Investment Strategy

in random walk (i.e., state increase). Particularly, if $\mathbf{E}_m^{s_t}(s_{t+1}) = s_t$, the player will fluctuate around this state and therefore stay at the same point in the long run. This is the *stable point* of such strategy, or the expected upper bound performance.

Furthermore, consider the *constant-investment strategy* that fixes the investment to a constant value disregard of the game state. In other words, it is the mini-batch SGD. Formally, we can obtain define an equation $\mathbf{E}_m^{s_t}(s_{t+1}) - s_t = 0$ for a particular constant investment m. If there is a solution for s_t, then the learning algorithm would fluctuate around this stable point. This is the potential limitation of mini-batch SGD.

5 BATCH-ADAPTIVE SGD

Section 4 proposed a random walk game to model the decrease of objective value with SGD learning. In this game, a larger sample size decreases the variance of random Gaussian walk, thus lowering the probability of objective value increase. Intuitively, one may use maximum sample size (the whole data set) in each iteration so that the variance is minimized. This is what Gradient Descent does and it is time consuming. To obtain a good balance of variance and time, one may need to choose proper sample size for each iteration. In this section we consider the following problem: given a limited budget (total number of samples), what is the best strategy of quota allocation to achieve minimum expected ending state.

Suppose we have a function that outputs the lowest achievable state given current state and remaining budget. Let $\mathtt{Q}(s_t, M)$ denote the function, we can easily write it in such a recursive form as Equation (9).

$$\mathtt{Q}(s_t, M) = \min\{\mathtt{Q}(\mathbf{E}_m^{s_t}(s_{t+1}), M - m) | m = 1, 2, \cdots, M\} \quad (9)$$

One may consider Dynamic Programming (DP) to solve the function. However, it is not realistic in SGD learning. Firstly, running DP requires an explicit knowledge of future state parameters $\langle \mu_{t+i}, \sigma_{t+i} \rangle$, which is unknown. Furthermore, even if we estimate the parameters, running one DP has a high time cost, especially when the M is large. Finally, optimal strategy obtained via DP is only for expected situations. That means, after each model update, original optimal strategy may not be valid any more and DP has to be run again. This is not practical. In this section, we provide a heuristic strategy that maximizes the efficiency of decreasing objective value. Then we discuss how the BA-SGD is developed based on this strategy.

5.1 MaxDec-Efficiency Strategy

With Equation(7) we can clearly calculate the expected value of next state. Since it is a monotonic decreasing function of investment, one straightforward strategy is to invest all budget in the first round to maximize the one-step descent of objective value. Alternatively, one may only make necessary investment to decrease the objective value and save the most for future. Table 2 shows a scenario of game setting and the strategy I and II are corresponding to such two intuitions.

Table 2: Example of Different Strategies

Game setting	$\mu_t = s_t, \sigma_t \equiv 1, \eta = 0.1, S^* = 0, M = 7$					
$s_0 = 0.1$	Round 1		Round 2		Round 3	
	m_1	$\mathbf{E}_{m_1}^{s_0}(s_1)$	m_2	$\mathbf{E}_{m_2}^{s_1}(s_2)$	m_3	$\mathbf{E}_{m_3}^{s_2}(s_3)$
I	7	0.0902	-	-	-	-
II	2	0.0968	2	0.0946	2	0.0930*
III	3	0.09296	4	0.0856	-	-

* The remaining of 1 budget can not make the state decrease based on Equation (7). So it stops at 3rd round.

As can be seen, the strategy I and II ends with state 0.0902 and 0.0930 respectively. Neither of them is better than strategy III, ending with state 0.0856. The key point is to consider the efficiency of investment, i.e., the decrease of state per each investment unit. Specifically, we can define the following utility function based on an investment and current state s.

$$\mathrm{u}(m, s_t) = \frac{s_t - \mathbf{E}_m^{s_t}(s_{t+1})}{m} \quad (10)$$

For each round, the most efficient investment is the one that maximize this utility function, i.e., $m_{\text{opt}} = \arg\max_m \mathrm{u}(m, s_t)$. Particularly for the first round, we show the state decrease and utility value with respect to different investment m in Figure 4. As can be seen, larger investment achieves bigger state decrease. However, the efficiency (utility value according to Equation (10)) only grows from 1 to 3 and then decreases. In that case, for first round, $m = 3$ is the one that achieves maximum return of unit investment. In the example shown in Table 2, the max-efficiency strategy gives an allocation of 3 and 4 for two rounds, i.e., the strategy III, named as *MaxDec-efficiency Strategy*.

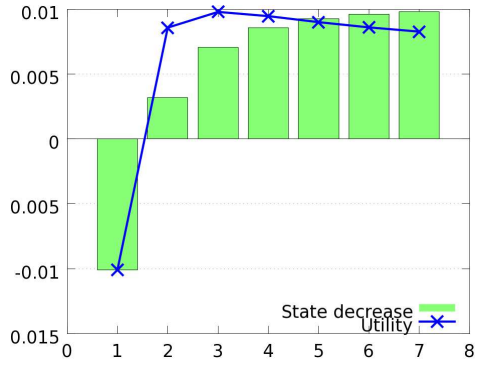


Figure 4: State decrease & Utility of different investment m

5.2 Algorithm Design

We show the pseudo code of BA-SGD in Algorithm 1. Besides common input such as data $\mathcal{X}$, initial model parameter $\vec{\theta}$, learning rate η, the algorithm requires two additional values, i.e., total budget M and sampling step m_0. The first one controls when the learning stops and the second determines the initial and incremental batch size. The algorithm starts with initial sample to estimate the derivative (line 7) and further the game state s_t, μ_t, σ_t (line 8). Then these estimated game parameters are used to calculate the batch size with MaxDec-efficiency strategy (line 9). Note that we arbitrarily set the minimum point S^* as 0. The real value of S^* depends on both data set and adopted objective function. Here we use a universal lower bound *0* as a safe choice because widely used objective functions, such as squared error, cross-entropy, etc., are usually non-negative. Sampling stops when the current sample is sufficient or if the whole data set is sampled (which makes the learning to gradient descent). Then the model parameter $\vec{\theta}$ and the remaining budget M are updated. The training stops when budget runs out.

Algorithm 1 Batch-Adaptive SGD

1: **procedure** BA-SGD($\mathcal{X}, \vec{\theta}, \eta, M, m_0$)
2: **while** $M > 0$ **do**
3: $\mathcal{Y} \leftarrow \emptyset$
4: **repeat**
5: random sample $\mathcal{Z}$ from $\mathcal{X} - \mathcal{Y}$ with $|\mathcal{Z}| = m_0$
6: $\mathcal{Y} \leftarrow \mathcal{Y} \bigcup \mathcal{Z}$
7: calculate $\hat{\mathrm{f}}_{\vec{\theta}}, \hat{\Sigma}$ with $\mathcal{Y}$ ▷ Equation (2) and (5)
8: $s_t \leftarrow \mathrm{F}(\vec{\theta}|\mathcal{Y}), \mu_t \leftarrow \hat{\mathrm{f}}_{\vec{\theta}}^T \hat{\mathrm{f}}_{\vec{\theta}}, \sigma_t \leftarrow \sqrt{\hat{\mathrm{f}}_{\vec{\theta}}^T \hat{\Sigma} \hat{\mathrm{f}}_{\vec{\theta}} / |\mathcal{Y}|}$
9: $m^* \leftarrow \arg\max_m \mathrm{u}(m, s_t)$ where $S^* = 0$
10: **until** $|\mathcal{Y}| \geq \min\{m^*, |\mathcal{X}|\}$
11: $\vec{\theta} \leftarrow \vec{\theta} - \eta \cdot \hat{\mathrm{f}}_{\vec{\theta}}$
12: $M \leftarrow M - |\mathcal{Y}|$
13: **end while**
14: **return** $\vec{\theta}$
15: **end procedure**

Particularly, to avoid redundant computation, the approximation of derivative and covariance matrix (line 7) can be done in a sequential manner, keeping record of the sum and self-multiplication of each individual derivative. Formally, let $\mathcal{Y}_1$ and $\mathcal{Y}_2$ denote the old and new sample, derivatives and covariance can be estimated by the following equation.

$$\hat{\mathrm{f}}_{\vec{\theta}} = \frac{\sum_{\mathrm{y}_j \in \mathcal{Y}_1 \cup \mathcal{Y}_2} \mathrm{g}_{\vec{\theta}}^j}{|\mathcal{Y}_1| + |\mathcal{Y}_2|} = \frac{\sum_{\mathrm{y}_j \in \mathcal{Y}_1} \mathrm{g}_{\vec{\theta}}^j + \sum_{\mathrm{y}_k \in \mathcal{Y}_2} \mathrm{g}_{\vec{\theta}}^k}{|\mathcal{Y}_1| + |\mathcal{Y}_2|}$$
$$\hat{\Sigma} = \frac{\sum_{\mathrm{y}_j \in \mathcal{Y}_1 \cup \mathcal{Y}_2} \mathrm{g}_{\vec{\theta}}^{j\,T} \mathrm{g}_{\vec{\theta}}^j - (|\mathcal{Y}_1| + |\mathcal{Y}_2|)\hat{\mathrm{f}}_{\vec{\theta}}^T \hat{\mathrm{f}}_{\vec{\theta}}}{|\mathcal{Y}_1| + |\mathcal{Y}_2| - 1} \tag{11}$$

One may note that the BA-SGD requires extra space of $O(d^2)$ to store the covariance matrix, where d is the dimension of the model parameter vector. For complex model, e.g., deep neural network, such cost is quite high. Furthermore, the calculation of the covariance matrix takes more time with larger d. A practical tradeoff is assume that model parameters are independent of each other. The covariance matrix is then a diagonal one, reducing the space cost to $O(d)$.

6 EVALUATION

To evaluate the proposed method, we run experiments on multiple data sets and compare its performance with other baselines, including conventional gradient descent, SGD algorithm, SGD algorithm with fixed learning rate (denoted as *Fix-SGD*), mini-batch SGD specifically with batch size 50 (*SGD-50*) and 100 (*SGD-100*).

6.1 Experiment Setup

The data sets we use in evaluation come from the published ones in UCI machine learning repository [21]. Particularly we look for clearly-described data sets of classification which i) has more than 10,000 instances, ii) has similar number of instances for each class and iii) has small number of features w.r.t data size. The second condition is to avoid imbalanced class problem and the third one is to avoid sparse feature problem as they are out of this work's scope. We end with 9 data sets, denoted as *Adult*, *Bank*, *Connect-4*, *Cover Type*, *Credit*, *Letter Recognition*, *Sensor*, *Skin* and *Teacher*. The summary is shown in Table 3.

By default we use neural network with one input layer, one output layer and 2 hidden layers. The input layer has the same number of neurons with the data's feature dimension. There are 5 neurons in both hidden layers. The neuron number in output layer is one for binary classification and is equivalent to the number of classes for multi-class classification. Experiments are run on a single Windows-7(64 bit) machine with dual CPU 2.9 GHz and 8 GB memory with Java Running Environment 1.8.0_65.

6.2 Model Verification

In previous sections we have modeled the stochastic gradient descent as well as the learning process. Here we use the real data to verify some of their characteristics. The data set *Adult* is used for illustration although we can get similar results in all data sets.

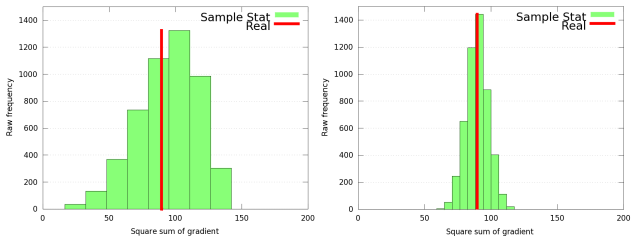
6.2.1 Gaussian Distribution of Gradient's Square Sum. In the model of stochastic gradient, we claim that its square sum $(\hat{\mathrm{f}}'_{\theta_0} \cdot \hat{\mathrm{f}}_{\theta_0})$

Table 3: Summary of Data Sets

Data	Class	Feature*	size	Name in UIC Repo
Adult	2	7	48,842	Adult
Bank [24]	2	24	41,188	Bank Marketing Data Set
Connect-4	3	42	67,557	Connect-4
Cover Type	7	14	581,012	Covertype
Credit [36]	2	23	30,000	default of credit card clients
Letter Recog	26	16	20,000	Letter Recognition
Sensor	11	48	58,509	Dataset for Sensorless Drive Diagnosis
Skin [3]	2	3	100,000+	Skin Segmentation
Teacher	4	16	10,800	Firm-Teacher_Clave-Direction_Classification

*Feature dimension may be different from raw data since we unfold categorical attributes and delete some which have too many values but for each value there is only a few number of instances.
+We remove some duplicate synthetic data records.

satisfies a Gaussian distribution on the basis of central limit theorem. Here we verify it by calculating stochastic ones on multiple sampling with varied sample size. Firstly, we check the raw frequency of sampled values. Specifically, with a neural network of the same configuration, we pick a random batch of a fixed size (e.g., 10 or 100) and calculate its gradient. This process is repeated 5000 times and the frequency is counted for each value.

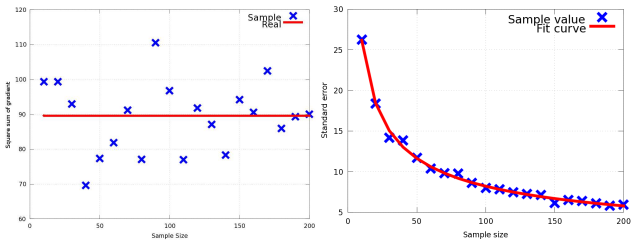


(a) Sample size 10 **(b) Sample size 100**

Figure 5: Raw Frequency

Figure 5 shows the histogram of sample gradient with two sample size. From the statistics of raw frequency count, they look similar to normal distribution. In the figure, we can easily see that the sample value distribute around a red line, which indicates the real value on global data. Comparing the two figures with sample size 10 and 100 respectively, we observe that the former one has a wider range of value, spanning from 20 to 150. The latter one has a much narrower range, from only 50 to 120. This scenario empirically confirms the theory that larger sample size reduces the noise variance.

In next experiment, we sample instances from the data set with different sample sizes only once and calculate the average of the individual sample gradient. Then the estimated squared sum of gradient is computed. The result is plotted in Figure 6a. Again the red line shows the real value calculated on the whole data. As can be seen, sample stats (blue crossings) disperse randomly along the red line. Also, for smaller sample size, it is more likely the sampled value is far from the real one, suggesting its higher variance.



(a) Average **(b) Standard Error**

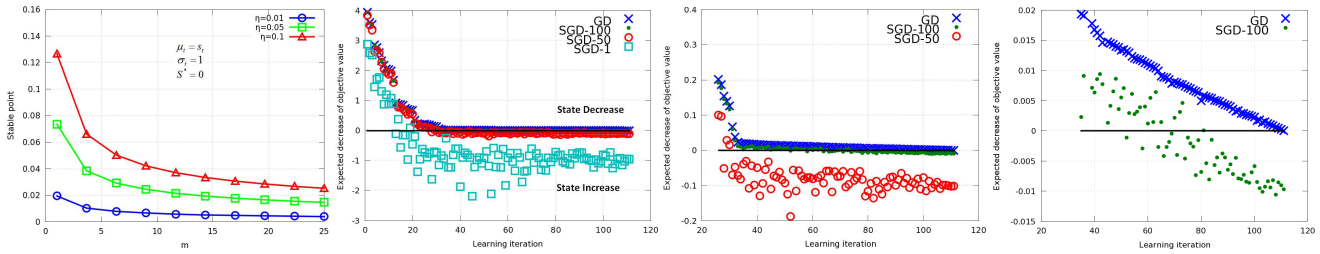
Figure 6: Impact of Sample Size

To further verify the hypothesis of decreasing variance, we repeat the sampling 50 times and calculate the standard error of the sample gradient with regarding to different sample size. With these pairs of sample size (x) and standard error (y), We fit a curve in the form of $y \sim \frac{1}{\sqrt{x}}$. The result is shown in Figure 6b. It can be easily seen that the standard error of sample (blue crossings) decreases as the sample size increases. Also, these points closely follow the fitted curve, demonstrating a decreasing trend proportional to the reverse of sample size's square root.

6.2.2 Stable Point of SGD. Recall in modeling of learning process we claim that there may be *stable point* for constant investment strategy. In simulated game we can explicitly obtain it by solving the equation. Particularly we set $S^* = 0$, and for all states s_t, $\mu_t \equiv s_t, \sigma_t \equiv 1$. Under such game design, the correlation between stable point and the choice of batch size m is shown in Figure 7a. As can be seen from the figure, for a fixed learning rate η, the stable point is decreasing with the increase of m. For the same value of m, on the other hand, the stable point is positively associated with the learning rate η. These two observations reveal that for unlimited rounds, bigger investment with smaller learning rate will always end with better performance.

In real learning, we show the stable point via an indirect way. Specifically we run the gradient-descent algorithm in the whole data set. For each iteration, before updating the model with whole data set, we run SGD algorithm to update a copy of the current model and calculate the resulted decrease of objective value. This process is repeated several times and the average decrease is recorded. The positive value indicates the adoption of the SGD will make the objective value decrease and the learning will move forward, vice versa. As the learning proceeds, we can estimate the potential performance of SGD algorithm at different objective value levels.

We display the results in Figure 7, where Figure 7b is the original figure and Figure 7c and 7d are the zoomed-in of the original one. The x-axis represents the learning iterations of gradient descent (GD), where larger value indicates lower objective value. The y-axis represents the decrease of objective value for a single iteration. There are mainly two observations. Firstly, in all three figures we can see that from some point the SGD-based algorithms have negative decrease (below the black horizontal line), indicating the objective value would increase if such learning algorithm were



(a) Simulation (b) Original (c) Zoom-in (d) Further Zoom-in

Figure 7: Verification of Stable Point for SGD. The x-axis represents the learning iterations of gradient descent (GD). The y-axis represents the decrease of objective value for a single iteration.

adopted. This observation confirms the existence of stable point for SGD with constant batch size. Secondly, by comparing the three figures we can see that the negative value of SGD-100 appears later than SGD-50 and SGD-1. It thus suggests the larger batch size has a smaller stable point. This observation confirms the simulation result in Figure 7a.

6.3 General Comparison

In this experiment, we evaluate the optimal value each learning algorithm achieves on the same data set, given the limited quantity of data instance. Due to different sizes of data sets, we use the concept of *epoch* as the measure of the quantity. Given a data set, one epoch represents the instance number equivalent to the data size. Similarly, two epochs stand for twice the total instance number and 0.2 epoch means 20% of the whole set. Furthermore, to know exactly the real cost value during learning, we run experiments in a "monitoring mode". That means, for SGD-based algorithms, we temporarily stop the learning process and compute the cost value on the whole data set as learning proceeds.

With such monitored training logs, we evaluate different algorithms in terms of two metrics, i.e., *descent efficiency* and *update efficiency*, defined in the equation below.

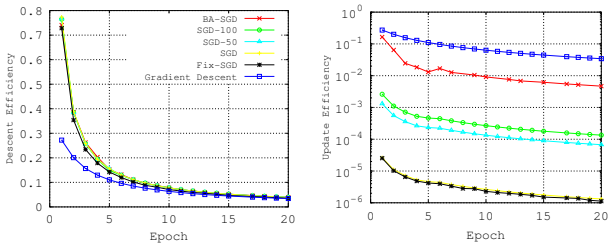
$$\text{descent efficiency} = \frac{1 - \frac{c_t}{c_0}}{N_t}$$
$$\text{update efficiency} = \frac{1 - \frac{c_t}{c_0}}{t} \tag{12}$$

where c_0 and c_t denote the objective value at beginning and at t^{th} iteration, N_t stands for the scanned data (in units of *epoch*) so far.

The first one is defined as the descent of the objective value over quantity of data scanned in training. It measures how efficient the learning algorithm makes use of training data. The higher the value is, the better the algorithm will be. The gradient descent (GD) algorithm is the lower bound since it scans the whole data to decrease the objective value in a single iteration.

The second metric evaluates how efficient the algorithm updates the model. Note that the number of iteration is equivalent to the number of model parameter updates. This metric is important for update-costly environment. For instance, in the parallel framework, updated model parameters need to be distributed to all peer machines for the next learning iteration. If update is too frequent, there will be too much communication burden for model synchronization, which leads to a long delay. At this metric, again a larger value indicates higher efficiency. Obiously the gradient descent algorithm is the upper bound as it requires minimum model update during learning.

Figure 8a shows the descent efficiency as learning proceeds. As can be seen, compared to the slow learning speed of gradient descent method, the SGD-based ones are much faster. This observation makes sense. The conventional gradient descent method update the model once scanning the whole data set while SGD method does with only a small sample. Therefore the latter one decrease quickly in an early stage. Note that our method curve overlaps with other SGDs', demonstrating its equivalent efficiency in learning.



(a) Descent Efficiency (b) Update Efficiency

Figure 8: General Comparison

Figure 8b shows the update efficiency of different learning algorithms with regarding to epochs. In general, the update efficiency shows a decreasing trend. This is normal. As learning proceeds, the model is closer and closer to optimum setting, making the margin of objective value decrease smaller and smaller. Thus the update efficiency is reduced. Also, we observe that all SGD-based method except ours decrease sharply. This observation is due to the high frequency of update and little improvement of the model. Moreover, the mini-batch SGD (SGD-50 and SGD-100) has a higher efficiency than SGD because update based on a batch instead of a single data sample can reduce the update frequency. The conventional gradient descent achieves highest efficiency because of its low number of update and slow decrease of objective value. Our BA-SGD falls between the two. On one hand, unlike other SGD methods, BA-SGD makes wise choice of when to update the model, depending

on the gradient's significance on the current sample. On the other hand, since BA-SGD has a faster convergence speed, as shown in Figure 8a, the efficiency is lower than gradient descent.

To sum up, in this experiment we demonstrate that our BA-SGD learning algorithm achieves equivalent convergence rate as other SGD methods, which is much faster than conventional gradient descent method. On the other hand, the BA-SGD has a relatively higher update efficiency than other SGD-based algorithm, making it rather competitive in parallel learning framework where the cost of model update and synchronization is quite high.

For specific data set, we show the final objective value of different learning methods in Table 4 after training with 20 epoch as budget. As can be seen, scanning the same amount of data instances, the gradient descent algorithm has highest objective value in all data sets, suggesting its slow learning speed compared to SGD-based algorithms. Also, our BA-SGD algorithm achieves lowest objective value in 7 data sets. For other 2, except skin, its objective value is close to the one with best performance. The dataset skin seems to be a different case, where the Fix-SGD achieves significantly lower objective value than other SGD-based ones. This dataset contains randomly generated data for non-skin classes. This synthetic method may result in low noise in the data, making the Fix-SGD rather efficient in learning.

6.4 Impact of Noise

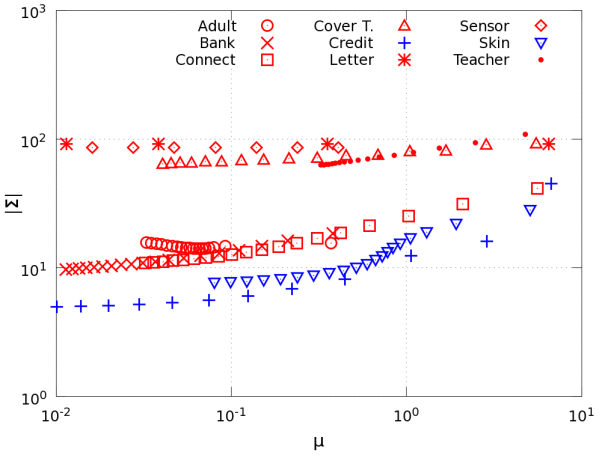


Figure 9: Noise Demonstration in Real Data

To further illustrate the impact of noise on the learning performance, we run gradient descent learning on all data sets and for each iteration log the value of derivative squared sum $\mu = \mathrm{f}_{\vec{\theta}}^T \cdot \mathrm{f}_{\vec{\theta}}$ as well as the determinant of the covariance matrix $|\Sigma|$. The pair of such values are plot in Figure 9. The x-axis is $\mathrm{f}_{\vec{\theta}}^T \cdot \mathrm{f}_{\vec{\theta}}$ while the y-axis is $|\Sigma|$. That means higher position corresponds larger scale of derivative noise. Particularly we use red to label data where BA-SGD achieves best performance as in Table 4 and blue for others. As can be easily seen, the blue markers (i.e., Credit and Skin) are below red ones.

We also run simulation of random walk game controlling the value of σ for the Gaussian dice. Compared strategies include constant investment with $m \equiv 1$, 5 & 10 and the MaxDec-Efficiency. The result is shown in Figure 10. As can be seen, given the same budget quantity, strategies of smaller constant investment reaches lower state when game ends. As noise scale increases, strategies of bigger constant investment performs better. Also, the MaxDec-Efficiency strategy is best among all cases. This is not consistent with results on real data. The reason is that in real learning scenario, the BA-SGD needs to have an initial sample to estimate statistics such as objective value, derivative and covariance matrix. In random walk game simulation, however, this information is explicitly and accurately given. Such initial sample (e.g., in experiment we set to 50) has to be larger than 1, making it less efficient than Fix-SGD algorithm when noise scale is small in the data set.

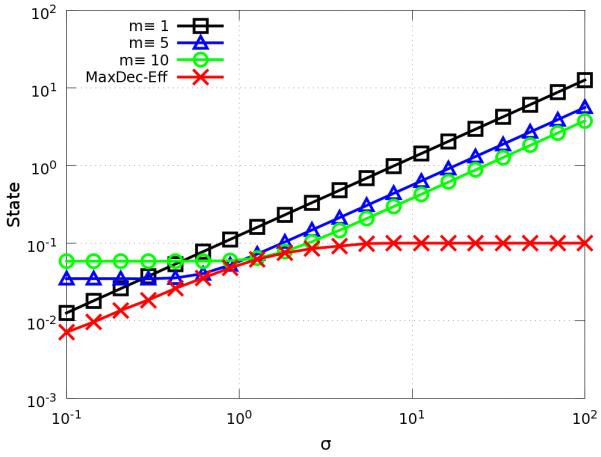


Figure 10: Impact of Noise in Simulation

7 CONCLUSION AND FUTURE WORK

In this work we develop a batch-adaptive SGD (BA-SGD) learning algorithm which can dynamically determine the batch size during each learning iteration. With support of the central limit theorem and statistic confidence interval, the algorithm has a good estimation of the risk when approximating gradient and can determine whether it needs more sample to reduce variance or proceed to update the model. As experiment shows, the BA-SGD achieves a convergence rate that is as fast as other SGD-based algorithms. At the same, it is more robust against noise.

BA-SGD algorithm provides an example of combining statistic theory with machine learning practice, worth further exploring. Also, there is quite a few arbitrary choice in the current version (e.g., initial sample size), which can be further improved by some advanced estimation techniques. we plan to study it in future work.

ACKNOWLEDGMENTS

Ping Luo is Supported by the National Natural Science Foundation of China (No. 61473274).

Table 4: Objective Value

	Data								
	Adult	Bank	Connect-4	Cover Type	Credit	Letter Rec.	Sensor	Skin	Teacher
BA-SGD	**0.5040**	**0.3534**	**0.3116**	**2.0070**	0.1683	**3.3375**	**1.9249**	0.1464	**0.9084**
SGD-100	0.5404	0.3562	0.4708	2.0773	0.1658	3.7909	3.3680	0.1469	1.3659
SGD-50	0.5378	0.3557	0.4681	2.0639	0.1651	3.7976	3.3246	0.1451	1.3556
SGD	0.5795	0.3543	0.4713	2.3864	0.1628	3.8857	3.6366	0.1433	1.7368
Fix-SGD	0.6624	0.4190	0.4339	6.0445	**0.1577**	4.1610	3.2110	**0.0030**	3.8869
Gradient Descent	0.9569	0.7078	0.7916	4.2286	0.3700	10.9911	5.8926	0.5546	2.9549

REFERENCES

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J Smola. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 123–132. ACM, 2012.

[3] Rajen Bhatt and Abhinav Dhall. Skin segmentation dataset. UCI Machine Learning Repository.

[4] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[6] Caihua Chen, Bingsheng He, Yinyu Ye, and Xiaoming Yuan. The direct extension of admm for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79, 2016.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam, 1996.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[11] The Apache Software Foundation. Apache hadoop. http://hadoop.apache.org/core/, 2009.

[12] The Apache Software Foundation. Mahout project. http://mahout.apache.org, 2012.

[13] Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976.

[14] Kevin Gimpel, Dipanjan Das, and Noah A Smith. Distributed asynchronous online learning for natural language processing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 213–222. Association for Computational Linguistics, 2010.

[15] Roland Glowinski and A Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(2):41–76, 1975.

[16] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[17] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

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

[21] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[22] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[23] Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pages 2915–2923, 2014.

[24] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[25] Noboru Murata. A statistical study of on-line learning. *Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK*, pages 63–92, 1998.

[26] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

[27] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[28] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[29] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[30] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[31] Shai Shalev-Shwartz and Tong Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 378–385, 2013.

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

[33] Evan R Sparks, Ameet Talwalkar, Valton Smith, Jey Kottalam, Xinghao Pan, Jose Gonzalez, Michael J Franklin, Michael I Jordan, and Tim Kraska. Mli: An api for distributed machine learning. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1187–1192. IEEE, 2013.

[34] Daniel Vainsencher, Han Liu, and Tong Zhang. Local smoothness in variance reduced optimization. In *Advances in Neural Information Processing Systems*, pages 2179–2187, 2015.

[35] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[36] I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[37] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[38] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[39] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.