Dhaka Univ. J. Sci. 64(2): 141-145, 2016 (July)

Minimizing Average of Loss Functions Using Gradient Descent and Stochastic Gradient Descent

Md. Rajib Arefin* **and M. Asadujjaman**

Department of Mathematics, Dhaka University, Dhaka-1000, Bangladesh

(Received: 29 December 2015; Accepted: 6 April 2016)

Abstract

This paper deals with minimizing average of loss functions using Gradient Descent (GD) and Stochastic Gradient Descent (SGD). We present these two algorithms for minimizing average of a large number of smooth convex functions. We provide some discussions on their complexity analysis, also illustrate the algorithms geometrically. At the end, we compare their performance through numerical experiments.

Keywords: Gradient Descent, Stochastic Gradient Descent, Convex Function, Unconstrained Optimization Problems.

I. Introduction

This paper presents two basic approaches for solving an unconstrained optimization problem of the form,

$$\min_{x\in\mathbb{R}^d} f(x) \tag{1}$$

where, $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

where $f_i : \mathbb{R}^d \to \mathbb{R}$, is a smooth convex function. Many problems in data science (e.g. machine learning, optimization and statistics) can be cast as loss minimization problems[3,4,7,1,6] of the form (1). There are several methods available for solving problem of the type (1). Some of these methods require second order derivatives of $f(x)$ while some other requires only first order derivatives. However, this paper will focus only on first order optimization algorithms, that is, the algorithms which do not require second order derivative of $f(x)$. We present two well-known first order algorithms namely, GD and SGD, for solving problem of the form (1). One of the oldest and well-known first order iterative algorithms for solving the above problem is GD that calculates the gradient of the function to have an update at each iteration. However, if the problem size becomes larger that is, if the number of functions included in the average is very large then minimizing average of those functions using GD would become very costly since it needs to evaluate the gradient of all the functions and consequently becomes less efficient. One remedy of this problem can be the use of SGD which, instead of considering all functions, estimates the gradient on the basis of a randomly chosen function in each iteration.

The rest of the paper is organized as follows. In section II, we discuss GD algorithm with some theoretical results on their complexity analysis and illustrate the algorithm geometrically. In section III, we discuss SGD algorithm with some theoretical results on their complexity analysis and illustrate it geometrically. Section IV, performs numerical experiments to compare the performance of GD and SGD. Finally, we draw a conclusion in section V.

II. Gradient Descent

Gradient descent[5] (also known as steepest descent) is an optimization technique for minimizing unconstrained multidimensional smooth convex function which starts with some initial parameters and performs a number of iterations (until expected accuracy is attained) towards the direction in which the value of the function decreases. Consider the problem (1), if $x = \hat{x}$ is a given point, $f(x)$ can be approximated by the linear expansion

$$f(\hat{x} + d) \approx f(\hat{x}) + \nabla f(\hat{x})^T d \tag{2}$$

Where d is small, i.e. $\|d\|$ is small. If the approximation in the expression (2) is good, then onewould want to choose d so that the inner product $\nabla f(\hat{x})^T d$ is as small as possible. It is easy to verify that the direction

$$\tilde{d} = -\frac{\nabla f(\hat{x})}{\|\nabla f(\hat{x})\|}$$

makes the smallest inner product with the gradient $\nabla f(\hat{x})$. This follows from the inequalities:

$$\nabla f(\hat{x})^T d \geq -\|\nabla f(\hat{x})\|\|d\| = \nabla f(\hat{x})^T \left(\frac{-\nabla f(\hat{x})}{\|\nabla f(\hat{x})\|}\right) = -\nabla f(\hat{x})^T \tilde{d}.$$

This is why, the un-normalized direction:

$$\tilde{d} = -\nabla f(\hat{x})$$

is called the direction of steepest descent at the point $\hat{x}$. Therefore, the idea is to take point x_j at iteration j, compute the gradient $\nabla f(x_j)$ and update

$$x_{j+1} = x_j - h_j \nabla f(x_j)$$

Where h_j is scalar step size. A natural consequence of this is the Algorithm of GD.

Existing Algorithm of GD

Step 1: **Input:** Initial point x_0, gradient norm tolerance ϵ.

Step 2: Set $j = 0$

Step 3: **while** $\|\nabla f(x_j)\| \geq \epsilon$ **do**

Step 4: $x_{j+1} = x_j - h_j \nabla f(x_j)$

* Author for correspondence. e-mail: arefin.math@du.ac.bd

Step 5: $j = j + 1$

Step 6: **end while**

Step 7: **Return:**x_j

Complexity Analysis of GD

To analyze the complexity of GD algorithm, we restrict to the case of convex-Lipschitz functions[2] as many problems lend themselves easily to this setting.

Definition 2.1

A differentiable convex function φ is said to have a *Lipschitz continuous gradient*, if there exists a constant $L > 0$, such that

$$\|\nabla\varphi(x) - \nabla\varphi(y)\| \leq L\,\|x - y\|, \quad \forall\, x, y$$

Definition 2.2

A convex function φ is *strongly convex* if and only if, there exist a constant $\mu > 0$ such that the function $\varphi(x) + \frac{\mu}{2}\|x\|^2$ is convex.

Different variants of gradient descent depend on how h_j is chosen. Smaller step size may lingerthe computational time (i.e. requires lots of iterations) whereas, larger step size can over shoot the minimum point, it may fail to converge, or even diverge. Choice of step sizes depends on the behaviour of the function. In addition to that it can give an estimation on the number of iterations needed. Note that, gradient descent can converge to a local minimum, even with the fixed step size. It is observed that as the iterates approach a local minimum, gradient descent will automatically take smaller steps. Therefore, no need to decrease the step size over time. The following results[9] gives an estimation of the number of iterations when the step size is constant.

Theorem 2.1

Suppose f has a Lipschitz continuous gradient with modulus L. Then the Algorithm of GD with a fixed step size $h_j = \frac{1}{L}$ will return a solution x_j with $\|\nabla f(x_j)\| \leq \epsilon$ in at most $O(1/\epsilon^2)$ iterations.

In addition to having a Lipschitz continuous gradient, if f is μ-strongly convex, then the rate of convergence can be strengthened which yields the following theorem.

Theorem 2.2

Consider the assumptions of Theorem 2.1. Moreover, assume that f is μ-strongly convex, and let $c = 1 - \frac{\mu}{L}$. Then $f(x_j) - f(x_*) \leq \epsilon$ after at most

$$\frac{\log((f(x_0) - f(x_*))/\epsilon}{\log(1/c)} \qquad (3)$$

iterations.

If GD is applied to a problem that is not strongly convex, it yields a low accuracy solution within a few iterations. However, as the iterations progress the algorithm stalls and no further increase in accuracy is obtained because of the $O(1/\epsilon^2)$ rates of convergence. On the other hand, if f is strongly convex, then GD converges linearly in $O(\log(1/\epsilon))$ iterations. It can be observed from the expression (3) that the number of iterations depends inversely on $\log(1/c)$. Approximating the denominator in (3) as $\log(1/c) = -\log(1 - \mu/L) \approx \mu/L$, hence, the algorithm requires atmost $O(\frac{L}{\mu}\log(1/\epsilon))$ iterations. Clearly, the convergence depends on the ratio $\kappa = L/\mu$. This ratiois called the condition number of a problem. If the problem is well conditioned, i.e., $\mu \approx L$ then GD converges extremely fast. Conversely, if $\mu << L$ then GD requires many iterations. Since GD calculates the gradient of n component functions per iteration, total complexity of the algorithm becomes $O(n\kappa\log(1/\epsilon))$.

Now if we do not want to use a constant step size for every iteration, then one can use the well known exact line search strategy[2] to get a proper step size. This strategy solves the following optimization problem for each iteration:

$$\min_{h>0} f(x_k - h\nabla f(x_k))$$

However, it is not always easy to solve this problem, specially when the function is non linear. In that case, inexact methods are commonly used in practice. These methods do not solve the problem exactly but estimate a good step size instead. However, we skip these methods as they are beyond the scope of this dissertation. Figure 2.1 illustrates the algorithm geometrically.

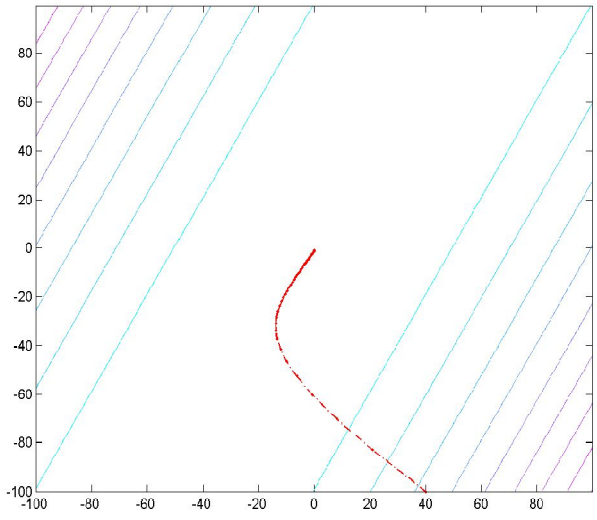

Fig.2.1 An illustration of the GD algorithm. The function to be minimized is $f(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(a_i^T x - b_i)^2$, where $x \in \mathbb{R}^2$ and $a_i \in \mathbb{R}^2$, $b_i \in \mathbb{R}$ are chosen arbitrarily.

In every iteration of GD, the iterates are pointing toward the greatest decrease of the function. Consequently, the algorithm shows a faster convergence. However, the algorithm becomes computationally expensive as the problem size increases. In that case, the well-known *Stochastic Gradient Descent (SGD)* is found to be more efficient than GD algorithm.

III. Stochastic Gradient Descent

The Stochastic Gradient Descent[3] (SGD) is a commonly used algorithm for minimizing an objective function that is written as the sum of differentiable functions such as, (1). It is drastic simplification of GD algorithm as it requires to compute the gradient of a single component function among all the admissible functions whereas, GD algorithm considers all the component functions to have a full gradient at each iteration. The idea of the algorithm is to choose a function randomly in every iteration and only compute the gradient of that function. Therefore, SGD picks a function f_i randomly from n component functions at each iteration and update

$$x_{j+1} = x_j - h_j \nabla f_i(x_j), \qquad i \in \{1, 2, 3, \dots, n\}$$

Clearly, this strategy reduces the amount of work drastically in each iteration because the complexity per iteration is now only 1. The following algorithm is a basic version of SGD.

Existing Algorithm of SGD

Step 1: **Input:** Initial point x_0, gradient norm tolerance ϵ.

Step 2: Set $j = 0$

Step 3: **while** $\|\nabla f(x_j)\| \geq \epsilon$ **do**

Step 4: Choose a function $f_i(x)$ randomly (uniform)

Step5: $x_{j+1} = x_j - h_j \nabla f_i(x_j)$

Step 6: $j = j + 1$

Step 7: **end while**

Step 8: **Return:** x_j

Since, $\mathbf{E}(\nabla f_i(x_j)) = \nabla f(x_j)$, we have an unbiased estimator of the gradient (full gradient). Therefore, the gradients of the component functions $f_1, f_2, \dots, f_n$ are referred to as stochastic gradients[12].

Complexity Analysis of SGD

The consecutive stochastic gradients may vary a lot or even point in opposite directions from the minimum point. Therefore, the convergence can be slow. The algorithm inherits the following convergence result[3]:

$$f(x_j) - f(x_*) \leq O(1/j) \text{ if } h_j = 1/j,$$

which depicts the slow convergence rate of SGD. Convergence results of SGD require step sizes satisfying the conditions $\sum_j h_j^2 < \infty$ and $\sum_j h_j = 0$. The convergence speed is in fact limited by the noisy approximation of the true gradient. When the step sizes decrease too slowly, the variance of the parameter estimate x_j decreases equally slowly. When the step sizes decrease too quickly, the expectation of the parameter estimate x_j takes a very long time to approach the optimum[3]. Usually iterates of SGD keep oscillating around the minimum of $f(x)$ but in practice, most of the values near the true minimum would be reasonably good approximations. Situations where low accuracy solutions are sufficient, i.e. small number of full gradient evaluations are sufficient to find an acceptable x, SGD works very well comparing to GD. That is why, SGD is extremely popularin the field of machine learning[6]. To be precise, in machine learning, usually a small number of passes over the data, which is actually the work equivalent to a small number of full gradient evaluations, are enough to get an acceptable solution.

In machine learning, the functions $f_i(x)$ in (1) are called loss functions which measure how well (or how poorly) the hypothesized function performs. The cost function $f(x)$ is then the average of the loss function. The computation of such a cost function takes a time proportional to the number of examples n. Since GD considers the complete cost function $f(x)$ at every iteration; as n becomesvery large, it would require a huge amount of efforts. The stochastic gradient instead updates the learning system, i.e. x_j on the basis of a single loss function. The algorithm works because the averaged effect of these updates is the same. Although the convergence can be much more noisy, the elimination of the constant n in the evaluation of cost function can be a huge advantage for large scale problems.

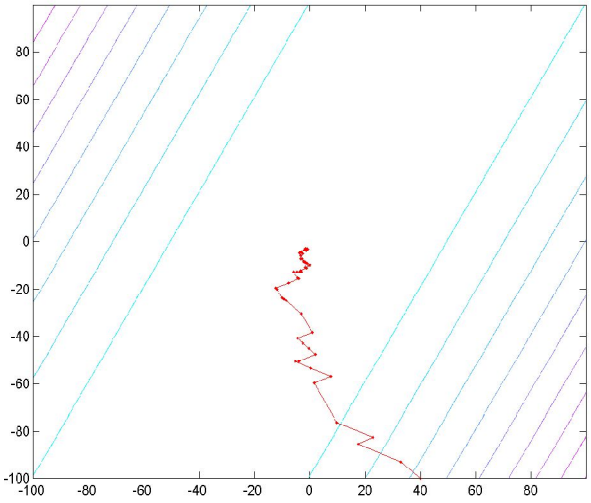

Fig.3.1 An illustration of the SGD algorithm. The function to be minimized is $f(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(a_i^T x - b_i\right)^2$, where $x \in \mathbb{R}^2$ and $a_i \in \mathbb{R}^2$, $b_i \in \mathbb{R}$ are chosen arbitrarily.

Figure 3.1 show how SGD behaves geometrically. The same function (as in GD) is used to see how the iterates of SGD converge to the minimum. Cleary the iterates are not always pointing toward the greatest decrease of the objective function, consequently, showing noisy updates.

Step Size Choice

Choosing the proper step size and changing its value over time can be challenging in SGD. In most SGD, the step size h is typically held constant. If we want the updates of

SGD to converge very close to the global minimum with less oscillation, we can slowly decrease the step size over time. A typical way to do that is to set the step size h as $\frac{a}{h+b}$, where a and b are two constants dictating the initial step size and time to change the step size respectively[10]. One needs to run the algorithm several times to determine these constants, therefore, takes more time than usual. However, if one can manage to have a proper estimation of these constants, SGD would give pretty good convergence to the global minimum. The more sophisticated method is to use back tracking line search method to find the optimal update. The following result[8] gives more intuition of the step size (constant) choice for SGD.

Theorem 3.1

Let $B > 0$ & Lipschitz constant, $L > 0$. Let f be a convex function and let $x^* \in \text{argmin}_{x:\, \|x\| \leq B} f(x)$. Assume that SGD is run for T iterations with $h = \sqrt{\frac{B^2}{L^2 T}}$. Assume also that for all j, $x_j \leq L$ with the probability 1. Then the output vector $\tilde{x}$ satisfies,

$$\mathbf{E}[f(\tilde{x}) - f(x^*)] \leq \frac{BL}{\sqrt{T}}$$

Therefore, for all $\epsilon > 0$, to achieve

$$\mathbf{E}[f(\tilde{x}) - f(x^*)] \leq \epsilon$$

it is sufficient to run the SGD algorithm for a number of iterations that satisfies,

$$T \geq \frac{B^2 L^2}{\epsilon^2}$$

A way to choose the variable step size is by setting $h_j = \frac{B}{L\sqrt{j}}$ to achieve a similar bound in Theorem 3.1. The idea is to choose the step sizes more carefully when we are closer to the minimum of the function so as not to "overshoot" the minimum point.

IV. Performance Comparison of GD and SGD

It is seen earlier that GD algorithm gives stable solution of a problem, that is, it can provide solutions to the expected accuracy. However, the algorithm is not efficient for the larger problems. SGD works well for the problems where solutions near the minimum is sufficient but it may not give more accurate solutions as the iterates of SGD usually oscillate around the minimum. Now let us consider a function of the form defined in (1), where $f_i(x) = \frac{1}{2}(a_i^T x - b_i)^2$. We choose an instance with $a_i \in \mathbb{R}^2$, $b_i \in \mathbb{R}$ as random and $n = 100$. Running both GD and SGD for several iterations gives the following figure which compares the performance of both algorithms.

As $n = 100$, running GD algorithm for a single iteration is equivalent to running SGD for 100 iterations. Moreover, one iteration of GD can be treated as a single pass over the data, therefore, running SGD for 100 iterations is equivalent to a single pass over the whole data set. Figure 4.1 represents a plot regrading the number of passes over data versus the difference between the functional value of the iterates and the optimum value.

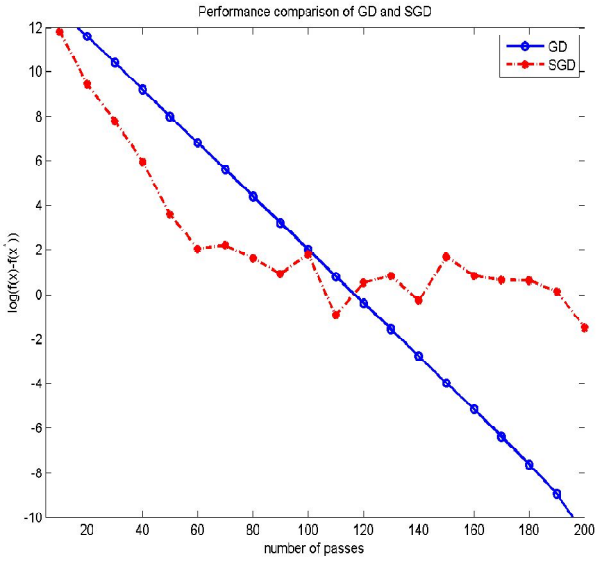

Fig. 4.1 Performance comparison of GD and SGD algorithms

The optimum value is evaluated using cvx (a MATLAB toolbox for convex optimization). We take the logarithm of the differences so that we can observe the differences in more narrow region. It is seen that SGD performs better than GD up to certain passes. After that it converges very slowly. In fact it may not converge to the exact minimum perhaps oscillates near the minimum point. However, the difference between the functional values of GD iterates and optimum value is consistently decreasing which dictates the stability of the algorithm. Therefore, whenever low accuracy solutions are sufficient, SGD could be a good choice but if more accurate solutions are required GD would perform better. However, as the problem size increases GD becomes more costly.

V. Conclusions

Minimizing the average of a large number of smooth convex loss functions frequently arises in practice. Classical approaches for solving these optimization problems are the GD and SGD. However, GD becomes computationally expensive for large problems, on the other hand, SGD is computationally cheap but not stable. We describe both algorithms with some theoretical results on their convergence. Moreover, we perform numerical experiments to compare their performance. It is observed that SGD performs better than GD at the beginning but after some iterations the iterates start oscillating. On the other hand, iterates of GD consistently move towards the minimum. The loss function we considered here is a least square function. It would be interesting to examine the performance of these algorithms for L2-regularized least squares as well as for logistic regression.

References

1. Bams D., T. Lehnert, and C. Wolff, 2005. Loss functions in option valuation: A framework for model selection. CEPR Discussion Papers 4960.
2. Bertsekas D. P., 1999. Nonlinear Programming, 2nd edition. Athena Scientific, Belmont, MA.
3. Bottou L., 2012. Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade (2nd ed.), 421–436.
4. Buja A., W. Stuetzle, and Y. Shen, 2005. Loss functions for binary class probability estimation and classification: Structure and applications, manuscript, available at www-stat.wharton.upenn.edu/ buja.
5. Freund R. M., 2004. Lecture note on the steepest descent algorithm for unconstrained optimization and a bisection-line search method. Massachusetts Institute of Technology.
6. Konečný J. and P. Richtárik, 2013. Semi-stochastic gradient descent methods. arXiv:1312.1666v1.
7. Schmidt M. W., N. L. Roux, and F. R. Bach, 2013. Minimizing finite sums with the stochastic average gradient. CoRR, abs/1309.2388.
8. Shalev-Shwartz S. and S. Ben-David, 2014. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, New York, NY, USA.
9. https://www.coursehero.com/file/6810633/Optimization/
10. Lecture on Machine learning by Andrew Ng, https://class.coursera.org/ml-005/lecture/preview