The Frontier of SGD and Its Variants in Machine Learning

Juan Du*
New Research and Development Center of Hisense, Qingdao 266071, China
*dqxwpl@sina.com

Abstract. A Numerical optimization is a classical field in operation research and computer science, which has been widely used in areas such as physics and economics. Although optimization algorithms have achieved great success for plenty of applications, handling the big data in the best fashion possible is a very inspiring and demanding challenge in the artificial intelligence era. Stochastic gradient descent (SGD) is pretty simple but surprisingly, highly effective in machine learning models, such as support vector machine (SVM) and deep neural network (DNN). Theoretically, the performance of SGD for convex optimization is well understood. But, for the non-convex setting, which is very common for the machine learning problems, to obtain the theoretical guarantee for SGD and its variants is still a standing problem. In the paper, we do a survey about the SGD and its variants such as Momentum, ADAM and SVRG, differentiate their algorithms and applications and present some recent breakthrough and open problems.

1. Introduction
Optimization is an important tool for making scientific precise decision and analyzing practical physical systems. In the simplest case, a normal optimization problem consists of a real objective function and series of associated constraints. The target of the problem is finding the maximum or minimum solution of the objective function in the feasible domain of the constraints. Formally speaking, the classic expression of the optimization issue is as following.

$$\begin{align*}
\min_{x \in \mathbb{R}^d} & \quad f(x) \\
\text{s.t.} & \quad g_i(x) \leq 0, \forall i \in \mathcal{E}, \\
& \quad h_j(x) = 0, \forall j \in \mathcal{J},
\end{align*}$$

(1)

Here $f(x)$ is the objective function, $g_i(x)$ and $h_j(x)$ give the inequality and equality constraints respectively.

It is a long story of the development of algorithms to solve the problem. In 1947, Dantzig [1] offered the first practical algorithm for linear programming which is known as simplex algorithm. However, it takes exponential time in the worst case. Later, Leonid Khachiyan proved that ellipsoid method can solve the linear programming. However, this method does not perform well in practice. The first practical polynomial time algorithm called Karmarkar’s algorithm [2]. Actually, it is one of the interior point methods, which is a type of algorithms that can solve both linear and nonlinear convex optimization issues. There are two common ways to design an interior point solution. One is called barrier method and the other is called primal-dual interior point method.
Except the methods for general convex programming, for some specific settings, there are more effective algorithms. One useful setting is that the programming is unconstrained, in which the support domain of the variables is unconstrained. Although it is more simplified version for the original problem, it is very common in the machine learning problems. Gradient descent (GD), which is also known as full batch gradient descent algorithm, is an implemented algorithm for the unconstrained problems. The iteration method of GD is pretty simple, and it can be described as below expression.

$$x^{t+1} = x^t - \gamma \nabla f(x^t)$$  \hspace{1cm} (2)

Where $\gamma$ is an adequately chosen step length. The algorithm achieves linear convergence when the initial estimate $x^0$ is close enough to the optimum and the step length $\gamma$ is sufficiently small. Here, linear convergence means $\log(\frac{1}{\epsilon}) \sim t$ where $\epsilon$ represents the residual error. Developed from the Newton’s method and inversion of the Hessian techniques, some later methods offer better alternate choices. These methods use fewer iterations to converge, while the cost of each iteration is higher. The BFGS [3] method is just in the case. Although these second order gradient descent methods usually achieve quadratic convergence, the second order differential may not exist or is very difficult to obtain. There are some techniques to avoid computing a Hessian matrix exactly. One common method is quasi-Newton method like BFGS.

There are some variants of BFGS, such as L-BFGS method, to handle the memory limited issues. The real-world problems are too large to compute the full batch gradient descent each time. We should note that the real problem in machine learning has more specific characters. Usually, in the supervised learning setup, the objective function is the sum of $f_i(x)$ as follows.

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$  \hspace{1cm} (3)

Each summand function $f_i(x)$ usually has close thing to do with the $i$-th observation in the data set. Typically, $f_i(x)$ stands for the loss at $i$-th example and $f(x)$ is often called empirical risk. There are many machine learning problems has the form, such as

**SVM:** $f(w) = \lambda w^2 + \sum \max\{0, 1 - y_i w^T \Phi(x_i)\}$ for $\Phi(x_i) \in \mathbb{R}^d, y = \pm 1$ \hspace{1cm} (4)

**K-Means:** $f(w) = \sum_{i=1}^{k} \min_{w_j, w_k} (z_i - w_j)^2$ for $z_i \in \mathbb{R}^d, w_1, ... w_d \in \mathbb{R}^d$ \hspace{1cm} (5)

**Lasso:** $f(w) = \lambda |w|_1 + \sum (1 - y_i w^T \Phi(x_i))^2$ for $\Phi(x_i) \in \mathbb{R}^d, y = \pm 1$ \hspace{1cm} (6)

In this setting, the stochastic gradient descent algorithm performs much better. Instead of computing the gradient descent exactly, in each iteration, randomly pick one component $f_i(x)$ and do the following update:

$$x^{t+1} = x^t - \gamma^t \nabla f_i(x^t)$$  \hspace{1cm} (7)

Note that the index $i$ is randomly picked in each iteration. Thus, in expectation, the SGD has the same performance as GD. There are plenty of works studying the performance of SGD in several different settings. For the non-smooth objective functions, the convergence rate of SGD is $O(1/\varepsilon^2)$, and for the smooth functions, the convergence rate is $O(1/\varepsilon^4)$ [4][5]. The searching path of SGD is shown in Figure 1 for an example (the sinuous vector path is called zig-zagging).
Figure 1. The illustration of the searching path of SGD

(\(x^i\) is the optimal result of the \(i^{th}\) iteration)

Here is a detailed example.

[Task 1]: The data of the 4 samples of function \(F(X) = \sum_{j=1}^{n-1} \theta_j x_j + \theta_n (\forall X \in R^{(n-1)})\) are listed in the Table 1 as below, please find the best simulated \(F(X)\).

| \(i\) (Sample No.) | \(x_1^i\) | \(x_2^i\) | \(F^i(x_1^i, x_2^i)\) |
|---------------------|----------|----------|----------------|
| 1                   | 2        | 1        | 11             |
| 2                   | 2        | 2        | 16             |
| 3                   | 3        | 1        | 14             |
| 4                   | 3        | 2        | 19             |

Table 1. Sample Set 01

Solution with algorithm SGD can be summarized as following:

Posit that the target linear function is \(F(x_1, x_2) = \theta_1 x_1 + \theta_2 x_2 + \theta_3\).

If \(\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}, X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\), then \(F(x_1, x_2) = X^T \theta (\theta \in R^3, X \in R^3)\)

[Iteration1]: Select a sample randomly from the sample set with 4 samples.

Posit that the sample randomly selected is sample 1: \(X_{s1}^{1x3} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}\), \(F_{s1}^1 = 11\)

The deviation function is: \(V(\theta)^{1}_{1x1} = F_{s1}^1 - F(X_{s1}^{1}) = 11 - (2\theta_1 + \theta_2 + \theta_3)\)

The loss function (Target of the optimization) \(J(\theta)^{1}_{1x1} = \frac{1}{2} (11 - (2\theta_1 + \theta_2 + \theta_3))^2\)

The 1st order gradient of \(J(\theta)\): \(G(\theta)^{1}_{1x1} = - (X_{s1}^{1})^T V(\theta)^1 = \begin{pmatrix} 2 \theta_1 + \theta_2 + \theta_3 - 11 \\ 2 \theta_1 + \theta_2 + \theta_3 - 22 \end{pmatrix}\)

Posit that the initial value of \(\theta\) is \(\theta_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\), then the initial \(G(\theta)^1 = \begin{pmatrix} -22 \\ -11 \end{pmatrix}\).

The deviation function is: \(V(\theta)^1 = 11, J(\theta)^1 = 60.5\)

If \(\eta^1\) is set to be 0.1, then \(\theta^2 = \theta_1 - \eta^1 G(\theta)^1 = \begin{pmatrix} 2.2 \\ 1.1 \\ 1.1 \end{pmatrix}\)
[Iteration2]: Repeat the rule of [iteration1] but change the sample with another one randomly selected from the sample set.

Assume that the second sample got randomly is sample 2: \( X^2_{1 \times 3} = \begin{pmatrix} 2 & 2 & 1 \end{pmatrix} \), \( F^2_{1 \times 1} = 16 \).

The deviation function is: \( V(\theta)^2_{1 \times 1} = F^2_{2} - F(X^2_{2}) = 16 - (2\theta_1 + 2\theta_2 + \theta_3) = 8.3 \)

The loss function (Target of the optimization) \( J(\theta)^2_{1 \times 1} = \frac{1}{2} (16 - (2\theta_1 + 2\theta_2 + \theta_3))^2 = 34.45 \)

The gradient of \( J(\theta) \): \( G(\theta)^2_{2 \times 1} = -(X^2_{2})^T V(\theta)^2 = \begin{pmatrix} 4\theta_1 + 4\theta_2 + 2\theta_3 - 32 & \theta_1 + 2\theta_2 + \theta_3 - 16 \end{pmatrix} \begin{pmatrix} -16.6 \\ -16.6 \end{pmatrix} \)

If \( \eta^2 \) is set to be 0.01, \( \theta^3 = \theta^2 - \eta^2 G(\theta)^2 = \begin{pmatrix} 2.37 \\ 1.27 \\ 1.18 \end{pmatrix} \), then \( J(\theta)^3 = 9.91 \)

Use the similar way to do following iterations. \( J(\theta)^4 = 0.01 \) (when \( \eta^3 = 0.15 \)).

2. The Variants of SGD

In this part, we discuss the core object: variants of SGD. These variants make SGD more useful and can be applied in more settings. There are several techniques for the variants, such as mini-batch, momentum, ADAM and variance reduced method. Each of them has their own characteristics, advantages and shortages. Then we introduce them below.

**Mini-batch.** Actually, instead of performing a parameter update for each single training date, usually, we perform the stochastic gradient descent for every mini-batch of `n`-training data. Generally speaking, it updates in following way.

\[
X^{t+1} = X^t - \gamma \nabla \sum_{t=B} f_t(x^t)
\]

Where set B includes the training data in a mini-batch. This way deceases the differences of the parameter updates, bringing more stable convergence and better effect. Moreover, it uses the strongpoints of highly optimized matrix optimizations which can increase the efficiency of computing the gradient w.r.t. a mini-batch very well. Hence, mini-batch SGD is typically used in machine learning training process.

Take Task 1 as example. Dividing the 4 samples into batch 1 (sample 1 and sample 2) and batch 2 (sample 3 and sample 4), then the n (quantity) of training data for each batch is 2. For the course of solving the Task 1, the only difference between SGD and mini-batch is that the former uses only one sample while the later uses one batch in each iteration \( X^1_{1 \times 3} \) and \( F^1_{1 \times 1} \) should be changed to be \( X^1_{2 \times 3} = \begin{pmatrix} 2 & 1 & 1 \\ 2 & 2 & 1 \\ 3 & 1 & 1 \end{pmatrix} \), \( F^1_{2 \times 1} = \begin{pmatrix} 11 \\ 16 \end{pmatrix} \), \( X^2_{1 \times 3} \) and \( F^2_{1 \times 1} \) should be changed to be \( X^2_{2 \times 3} = \begin{pmatrix} 3 & 1 & 1 \\ 2 \end{pmatrix} \), \( F^2_{2 \times 1} = \begin{pmatrix} 14 \\ 19 \end{pmatrix} \), and there is no the course of random selection in Mini-batch.

**Momentum.** When the surface curves are much steeper in one dimension than that in other dimensions, SGD has poor performance since it would oscillate across the slopes. A simple way to overcome the weakness is to introduce a momentum term in the update iteration. Momentum simulates the concept inertia in physics. It means that in each iteration, the update mechanism is not only related to the gradient descent, which refers to the dynamic term, but also maintains a component which related to the direction of last update iteration, which refers to the momentum. Since it is similar to push a ball down in hill, the momentum method is also named heavy ball method. Formally, it has the following update rule:

\[
v^t = x^t - x^{t-1}, \quad v^{t+1} = \rho v^t - \gamma \nabla \sum_{t=B} f_t(x^t), \quad x^{t+1} = x^t + v^{t+1}
\]

Where \( \rho(x^t - x^{t-1}) \) is the momentum. The constant \( \rho \) determines the extent of inertia. The effect of Momentum can be shown in Figure 2 for an example.
Figure 2. The compare of the SGD algorithms with and without momentum

Take Task 1 as example. The algorithm in the end of the 3rd iteration of SGD should be changed from \( \theta^3 = \theta^2 - \eta^2, G(\theta)^2 = (2.37, 1.27) \) to \( \theta^3 = \theta^2 - \eta^2, G(\theta)^2 + \rho(\theta^2 - \theta^1) = (3.47, 1.82) \) (set \( \rho = 0.5 \)) for Momentum. It shows that the value \( \theta^3 \) approximates to the optimum faster with Momentum.

There are some other ways to choose momentum. Nesterov’s momentum is an improvement, which is known as Nesterov’s Accelerated Gradient (NAG) descent [6]. The complexity of NAG matches the theoretical lower bound which is \( O\left(\frac{1}{\sqrt{\varepsilon}}\right) \) for smooth functions and \( O\left(\log\left(\frac{1}{\varepsilon}\right)\right) \) for strongly convex and smooth functions. Start at an arbitrary initial point, NAG iterates the following equations for each step.

\[
v^t = (x^t - x^{t-1} + \rho v^{t-1})/(1 + \rho), v^{t+1} = \rho v^t - \nabla \sum_{i=1}^{n} f_i(x^t), x^{t+1} = x^t - \rho v^t + (1 + \rho)v^{t+1}
\]

(10)

There are some works to explain the intuition of NAG and study whether it is the best iterated rules. Recently, Wibisono and et al. [7] show that the Nesterov’s momentum has strong relation with the partial differential equations. They show that the trajectory of NAG is corresponding to brachistochrone according to the variational method. See more details for the Nesterov’s method, we refer reader to see the survey [8]. Recently, Allen-Zhu provides a new momentum named Katyusha momentum and achieves great performance in many settings [4].

An illustration of the difference among SGD, Momentum & Nesterov’s momentum is Figure 3:

Figure 3. An illustration of the difference among SGD, Momentum & Nesterov’s momentum

**ADAM methods.** So far, the method we introduced use the constant step length, which is related to the final computational accuracy. There are some methods to update the step length automatically, such as vSGD, Adadelta, AdaGrad, RMSProp, ADAM, NADAM [9][10][11]. AdaGrad can adapt the rate of learning according to the disparate relevant variables, and perform larger updates for the infrequent while smaller updates for those frequent parameters. Adadelta works particularly well for
sparse gradients. For RMSProp, it performs well for on-line and non-stationary configurations. Consequently, ADAM is devised to cover the advantages of both AdaGrad and RMSProp, which is not only efficient computationally, but also needs little memory and suits for problems that are large in terms of data or parameters very well. Then, ADAM’s update rule is as following
\( g^{t+1} = \nabla f(x^t), \quad m^{t+1} = \beta_1 m^t + (1 - \beta_1) g^{t+1}, \quad v^{t+1} = \beta_2 v^t + (1 - \beta_2) (g^{t+1})^2 \)
\( m^{t+1} = m^{t+1} / (1 - (\beta_1)^{t+1}), \quad v^{t+1} = v^{t+1} / (1 - (\beta_2)^{t+1}), \quad x^{t+1} = x^t - \gamma m^{t+1} / (\sqrt{v^{t+1}} + \epsilon) \)

ADAM is the best proposed algorithm for stochastic optimization and a good default choice for most cases. Show it in detail by taking Task 1 as example here. Set \( \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \theta^1 = 0, m^1 = 0, v^1 = 0, \eta^1 = 0.1 \) are the initial values, then the algorithm in the end of the 1st iteration of SGD should be changed from \( \theta^2 = \theta^1 - \eta^1 G(\theta)^1 \) to be following [New Iteration1] algorithm:
\[ g^2 = G(\theta)^1 = \begin{pmatrix} -22 \\ -11 \\ -11 \end{pmatrix}, \quad m^2 = \beta_1 m^1 + (1 - \beta_1) g^2 = \begin{pmatrix} -2.2 \\ -1.1 \\ -1.1 \end{pmatrix}, \quad v^2 = \beta_2 v^1 + (1 - \beta_2) (g^2)^2 = \begin{pmatrix} -11.58 \\ -5.79 \\ -5.79 \end{pmatrix}, \quad \theta^2 = 0.73, \quad m^{1/2} = \sqrt{m^2 / (1 - (\beta_2)^2)} = \begin{pmatrix} 1.36 \\ 0.68 \\ 0.68 \end{pmatrix} \]

For the following iterations with ADAM, the iterative rules are same. **SVRG.** Recently, some researchers find that SGD has slow convergence asymptotically due to the inherent variance. By now, Stochastic Variance Reduced Gradient (SVRG) [12] and its variants such as S2GD, SAG and SAGA [1][13] have delivered much progress through taking advantage of the variance reduced technique. SVRG has linear convergence results for smooth and strongly convex loss. At each time of SVRG, keep a version of estimated \( \bar{x} \) as that is close to the optimal. For example, keep a snapshot of \( \bar{x} \) after every \( m \) SGD iterations. Moreover, maintain the average gradient. \( \mu = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\bar{x}) \). Then, the update rule of SVRG is as below equation:
\( x^{t+1} = x^t - \gamma (\nabla f_i(x^t) - \nabla f_i(\bar{x}) + \mu) \)

Despite the meteoric rise of VR methods, their analysis for general non-convex problems is largely missing. Recently, there are some works studying to apply SVRG algorithms for the non-convex settings [14].

3. Distributed and Non-convex Extension
So far, we know that SGD is one of the hottest algorithms which achieve state-of-the-art performance for plenty of various machine learning issues. In the big data era, there are two principle challenges for SGD. One is to apply it in the distributed system. The other is to process the non-convex objective functions, which is common in machine learning problems.

**Distributed Extension.** Several researchers have suggested plans to parallelize SGD, one of the most famous one is Hogwild [15]. Actually, the algorithm is simply applying SGD in the distributed system without any locking. It means that the algorithm makes it possible for processors’ sharing memory with the likelihood of overwriting their work. For the case that the optimization problem is sparse, meaning most gradient updates merely change the decision variable in small part, therefore Hogwild realizes a nearly optimal convergence rate. Recently, there are some improvement for Hogwild Besides, there are some work applying the SGD in the dual problem. One of the typical algorithm is parallel dual coordinate descent algorithm [16].

**Non-convex Extension.** Although, it has achieved great process for the theoretical analysis for SGD recent years, most of the results can only apply for the convex objective functions. As we know, most of the objective functions in machine learning problems are non-convex and SGD works well in
practice. To achieve the theoretical guarantee for SGD is the current research focus. There are some partial progresses in this problem. Several works study some special non-convex objective functions and find SGD or its variants can be convergence. Besides, some researchers [17] find that in many machine learning problems, the minimal value of local minimum is a good approximation for the global minimum. Moreover, it is not difficult to obtain a local minimum since the quantity of local minimum is significant. Very recently, there are researchers studying the hitting time [18] for the optimization problems. They find that although, the mixing time (convergence to global minimum) is exponential long in some problems, the hitting time is polynomial. Hence, we can expect to get a good approximation results in polynomial time.

4. Conclusion
Optimization problem is the essential problem in many areas such as computer science, physics and economics. As we know, today is the big data era, and machine learning is playing more and more significant role on various aspects of life. How to handle the terabyte-class or even petabyte-class datasets perfectly is a great challenge for the recent academic researchers and industrial algorithm designers. In this paper, we introduce the start-of-art algorithm for unconstrained problems in finite-sum form to give a high efficient summary as well as try to enlighten the further development in this direction. Moreover, we explain the variants of SGD in details and discuss the two current frontiers about the SGD algorithms, so that the important idea on SGD can be better understood and applied in more fields.

Reference:
[1] Dantzig, George, Alexander Orden, and Philip Wolfe. “The generalized simplex method for minimizing a linear form under linear inequality restraints.” Pacific Journal of Mathematics 5.2 (1955): 183-195.
[2] Adler, I., Resende, M. G., Veiga, G., & Karmarkar, N. (1989). An implementation of Karmarkar's algorithm for linear programming. Mathematical programming, 44(1), 297-335.
[3] Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1), 503-528.
[4] Allen-Zhu, Z. (2016). Katyusha: The first direct acceleration of stochastic gradient methods. arXiv preprint arXiv:1603.05953.
[5] Bertsekas, D. P. (2011). Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. Optimization for Machine Learning, 2010(1-38), 3.
[6] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.),vol.269,pp.543–547
[7] Wibisono, A., Wilson, A. C., & Jordan, M. I. (2016). A variational perspective on accelerated methods in optimization. Proceedings of the National Academy of Sciences, 201614734.
[8] Nesterov, Y. (2013). Introductory lectures on convex optimization: A basic course (Vol. 87). Springer Science & Business Media.
[9] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning. University of Toronto, Tech. Rep.
[10] Defazio, A., Bach, F., & Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Advances in Neural Information Processing Systems (pp. 1646-1654).
[11] Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In Advances in neural information processing systems (pp. 315-323).
[12] Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In Acoustics, speech and signal processing (icassp), 2013 ieee international conference on (pp. 6645-6649). IEEE.
[13] Kingma, D., & Ba, J. (2014). ADAM: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
[14] Reddi, S. J., Hefny, A., Sra, S., Poczos, B., & Smola, A. (2016, June). Stochastic variance reduction for nonconvex optimization. In International conference on machine learning (pp. 314-323).

[15] Recht, B., Re, C., Wright, S., & Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in neural information processing systems (pp. 693-701).

[16] Tseng, P. (2001). Convergence of a block coordinate descent method for nondifferentiable minimization. Journal of optimization theory and applications, 109(3), 475-494.

[17] Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., & Ma, T. (2017, June). Finding approximate local minima faster than gradient descent. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (pp. 1195-1199). ACM.

[18] Zhang, Y., Liang, P., & Charikar, M. (2017). A Hitting Time Analysis of Stochastic Gradient Langevin Dynamics. arXiv preprint arXiv:1702.05575.