Stochastic Optimization for Machine Learning

Shuai Zheng
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
szhengac@cse.ust.hk
Supervisor : James T. Kwok

July, 2017

Abstract

Numerical optimization has played an important role in the evolution of machine learning, touching almost every aspect of the discipline. Stochastic approximation has evolved and expanded as one of the main streams of research in mathematical optimzation. This survey provides a review and summary on the stochastic optimization algorithms in the context of machine learning applications. The stochastic gradient descent (SGD) method has been widely viewed as an ideal approach for large-scale machine learning problems while the conventional batch gradient method typically falters. Despite its flexibility and scalability, the stochastic gradient is associated with high variance which impedes training. Based on this viewpoint, we provide a comprehensive theoretical and practical discussion on the SGD, and then we investigate a new spectrum of incremental gradient methods that suppress the noise in a smart way, leading to improved convergence results. We further present review on methods that integrate the stochastic gradients into the alternating direction method of multipliers (ADMM), which has been recently advocated as an efficient optimization tool for a wider variety of models. Last but not least, we also presents some stochastic optimization techniques for the deep neural network training, including momentum methods and algorithms with adaptive learning rates.

Contents

1 Introduction

In this big data era, tons of information are generated every day. Thus, efficient optimization tools are needed to solve the resultant large-scale machine learning problems. It has been long acknowledged that a batch optimization algorithm can minimize the objective at a fast rate. However, it suffers from high computational cost, as its per-iteration computing time is propotional to the number of training samples n. This turns our attention to the stochastic approximation methods that are more scalable for large-scale problems. In particular, the well-known stochastic gradient descent (SGD) [58, 8] and its variants [9] have drawn a lot of interest. Instead of visiting all the training samples in each iteration, the gradient is computed by using one sample or a small mini-batch of samples. The per-iteration complexity is then reduced from $O(n)$ to $O(1)$. Despite its scalability, the stochastic gradient is much noisier than the batch gradient. Thus, the stepsize has to be decreased gradually as stochastic learning proceeds, leading to slower convergence.

Recently, in convex optimization, there has been a lot of interest in reducing the variance of the stochastic gradient. A pioneering work is the stochastic average gradient (SAG) [59], which reuses the old stochastic gradients computed in previous iterations. With the variance reduced, a larger constant stepsize can be used. Under mild conditions, it has significantly better convergence than black-box optimization both theoretically and empirically. A related method is stochastic dual coordinate ascent (SDCA) [66], which performs stochastic coordinate ascent on the dual. However, a caveat of SAG is that storing the old gradients takes $O(nd)$ space, where d is the dimensionality of the model parameter. Similarly, SDCA requires storage of the dual variables, which scales as $O(n)$. Thus, they can be expensive in applications with large n (big sample size) and/or large d (high dimensionality). Stochastic variance reduced gradient (SVRG) [36] alleviates this storage problem by keeping snapshots of the full gradient and parameter estimator. Its extra memory requirement is low even when the sample size get very large. In light of its efficiency, SVRG has been extended to many scenarios [81, 56, 23, 86, 91, 88]. Very recently, motivated by the success of variance reduction in convex optimization, convergence results on nonconvex optimization have also been provided for SVRG [2, 57].

Moreover, many machine learning problems, such as graph-guided fused lasso and overlapping group lasso, are too complicated for SGD-based methods. The alternating direction method of multipliers (ADMM) has been recently advocated as an efficient optimization tool for a wider variety of models [13]. Stochastic ADMM extensions have also been proposed [52, 71, 77], though they only have suboptimal convergence rates. Recently, researchers have borrowed variance reduction techniques into ADMM. The resultant algorithms, SAG-ADMM [92] and SDCA-ADMM [72], have fast convergence rate as batch ADMM but are much more scalable. The downside is that they also inherit the storage drawback of SAG and SDCA. To alleviate this problem, one can integrate ADMM with SVRG, resulting in an algorithm [88] with lower memory cost.

Deep learning has emerged as a powerful and popular class of machine learning algorithms. Well-known examples include the convolutional neural network [42], long short term memory [33], memory network [78], and deep Q-network [47]. These models have achieved remarkable performance on various difficult tasks such as image classification [31], speech recognition [28], natural language understanding [5, 69], and game playing

[67]. Deep network is a highly nonlinear model with typically millions of parameters [32]. Thus, it is imperative to design scalable and effective solvers. However, training deep networks is difficult as the optimization can suffer from pathological curvature and get stuck in local minima [44]. Moreover, it has been shown that every critical point that is not a global minimum is a saddle point [38], which can significantly slow down training. Second-order information is useful in that it reflects local curvature of the error surface. However, a direct computation of the Hessian is computationally infeasible. Martens (2010) introduced Hessian-free optimization, a variant of truncated-Newton methods that relies on using the linear conjugate gradient to avoid computing the Hessian. Dauphin et al. (2014) proposed to use the absolute Hessian to escape from saddle points. However, these methods still require higher computational costs.

Recent advances in deep learning optimization focus mainly on SGD and its variants using momentum techniques [70]. However, SGD requires careful stepsize tuning, which is difficult as different weights have vastly different gradients (in terms of both magnitude and direction). On the other hand, online learning [93], which is closely related to stochastic optimization, has been extensively studied in the past decade. Well-known algorithms include follow the regularized leader (FTRL) [37], follow the proximally-regularized leader (FTPRL) [45] and their variants [21, 22, 64, 80]. In particular, adaptive gradient descent (Adagrad) [22] uses an adaptive per-coordinate stepsize. On convex problems, it has been shown both theoretically and empirically that Adagrad is especially efficient on high-dimensional data [22, 46]. When used on deep networks, Adagrad also demonstrates significantly better performance than SGD [16]. However, in Adagrad, the variance estimate underlying the adaptive stepsize is based on accumulating all past (squared) gradients. This becomes infinitesimally small as training proceeds. In more recent algorithms, such as RMSprop [74] and Adam [40], the variance is estimated by an exponentially decaying average of the squared gradients. Another problem with the FTRL family of algorithms is that in each round, the learner has to solve an optimization problem that considers the sum of all previous gradients. For highly nonconvex models such as the deep network, the parameter iterate may move from one local basin to another. Gradients that are due to samples in the distant past are less informative than those from the recent ones. In applications where the data distribution is changing (as in deep reinforcement learning), this may impede parameter adaptation to the environment. To alleviate this problem, a FTPRL variant, called FTML, is proposed in [90]. This method shows strong connections with deep learning optimizers such as RMSprop and Adam.

In this survey, we aim to give a comprehensive overview of stochastic algorithms for machine learning. The rest of this survey is organized as follows. Section 2 first gives a brief review on classic stochastic approximation algorithms. Section 3 then presents the incremental gradient methods. Section 4 covers the stochastic techniques for ADMM. Section 5 surveys the optimization methods for deep neural network training. The last section gives some concluding remarks and future research discussion.

2 Stochastic Approximation

Stochastic approximation (SA) is a type of computational approach based on Monte Carlo sampling techniques. The SA method is going back to the pioneering work by Robbins and Monro [58]. The stochastic gradient descent (SGD) has been widely known as one of the most ideal optimization approaches for large-scale machine applications due to its efficiency and scalability. In this section, we will provide an overview of the classic SA method - SGD, and discuss why stochastic methods have inherent advantages over batch methods for machine learning applications.

2.1 Problem Setup

Optimization problems in machine learning corresponds to the minimization of an objective that measures the quality of our predictions. It is often useful to minimize the population risk, which measures generalization performance [10, 11]. In the following, we consider minimization problems of the form:

$$\min_x F(x) := \mathbf{E}[f(x, \xi)], \tag{1}$$

where x is the model parameter in $\mathbb{R}^d$, f is a loss function, ξ is a sample drawn from some underlying distribution, and the expectation is taken w.r.t. ξ. For example, one may imagine a realization of ξ as a single example with input-output pair. Note that the underlying distribution of ξ can be (i) the true data distribution, (ii) the uniform distribution over a finite training set with random data perturbation (such as dropout noise), or simply (iii) the training set distribution. In the latter case, (1) reduces to minimizing the empirical loss.

In the sequel, to simplify the notation, let us define the loss incurred by the parameter vector x with respect to the sample drawn at round t as

$$f(x, \xi_t) = f_t(x).$$

To establish the convergence results, the following assumptions are used throughout this survey.

Assumption 1 *Each loss function f_i is convex on $\mathbb{R}^d$. Then*

$$f_i(x) \geq f_i(y) + \nabla f_i(y)^T (x - y)$$

for all $x, y \in \mathbb{R}^d$.

Note that any local minimum of a convex function is also a global minimum.

Assumption 2 *Each loss function f_i is continuously differentiable on $\mathbb{R}^d$, and has Lipschitz-continuous gradient. Hence, there exists $L > 0$ such that*

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$$

for all x, y.

Algorithm 1 Stochastic Gradient Descent (SGD).

Input: $\eta_t > 0$.
initialize $x_0 \in \mathbb{R}^d$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $g_t = \nabla f_t(x_{t-1})$;
  $x_t = x_{t-1} - \eta_t g_t$;
end for

This implies the average cost function F is also smooth, with

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x - y\|,$$

The smoothness ensures that the gradient provides a good indicator to measure the decrease on the objective.

Assumption 3 *Each loss f_i is strongly convex on $\mathbb{R}^d$, i.e., there exists $\mu > 0$ such that*

$$f_i(x) \geq f_i(y) + \nabla f_i(y)^T(x - y) + \frac{\mu}{2}\|x - y\|^2$$

for all $x, y \in \mathbb{R}^d$.

The Assumption 3 also implies that F is μ-strongly convex. If F is strongly convex, then there exists an unique minimizer to the objective, denoted as $x_* = \arg\min_x F(x)$.

2.2 Stochastic Gradient Descent

As the optimization objective in (1) is an expectation, it may be infeasible to compute the true gradient when the underlying distribution is the ground truth data distribution, as infinite samples will be needed. To approximate the true gradient, SGD uses a single gradient $\nabla f_t(x)$, where function f_t corresponds to the loss incurred by the sample drawn from the data distribution at time t. The complete SGD algorithm is presented in Algorithm 1. It can be easily seen that SGD is easy to implement, memory-efficient and has low per-iteration complexity. Since the SGD does not need to remember the past visited examples, it can perform update on the fly in a deployed system.

Indeed, $\nabla f_t(x)$ may not produce a descent direction for the objective. However, if it is a descent direction from x in expectation, then the iterate sequence can still be guided toward a minimizer of the objective. As the stochastic gradient is a noisy estimate for the true gradient $\nabla F(x)$, it will incur some variance. In stochastic optimization, its variance is usually assumed to be bounded by a constant as

$$\mathbf{E}[\|\nabla f_t(x) - \nabla F(x)\|^2] \leq \sigma^2$$

for any $x \in \mathbb{R}^d$. As shown in [12], for minimizing strongly convex functions, if the stepsize policy satisfies:

$$\eta_t = \frac{c}{\gamma + k} \text{ for some } c > \frac{1}{\mu} \text{ and } \gamma > 0 \text{ such that } \eta_1 \leq \frac{1}{L},$$

then, SGD achieves following convergence rate:

$$\mathbf{E}[F(x_t)] - F(x_*) \leq \frac{\nu}{\gamma + t + 1}, \tag{2}$$

where

$$\nu = \left\{ \frac{c^2 L \sigma^2}{2(c\mu - 1)}, (\gamma + 1)[F((x_0)) - F(x_*)] \right\}$$

The convergence result shows that SGD achieves a convergence rate of $O(1/t)$, which matches optimal asymptotic minimax rate for stochastic approximation [1, 49]. The restriction $c > \frac{1}{\mu}$ is critical to obtaining the optimal $O(1/t)$ rate. It has been observed that the underestimation of c can make the convergence extremely slow [50]. Here, the convergence result is established when η_t decreases in a rate of $O(1/t)$. In fact, the convergence of the SGD can be obtained with a more general stepsize sequence:

$$\sum_{t=1}^{\infty} \eta_t = \infty \text{ and } \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

When a constant stepsize is used, it is only guaranteed that the expected suboptimality gap converges near the zero. Specifically, if the constant stepsize η satisfies:

$$0 < \eta \leq \frac{1}{L},$$

then, the expected optimality gap satisfies the following:

$$\mathbf{E}[F(x_t)] - F(x_*) \leq \frac{\eta L \sigma^2}{2\mu} + (1 - \eta\mu)^t \left(F(x_0) - F(x_*) - \frac{\eta L \sigma^2}{2\mu} \right). \tag{3}$$

Thus, with a constant step size, SGD forgets the initial gap exponentially fast and converges linearly to a neighborhood of the solution where the noise in the gradient estimates prevent further progress. Besides, selecting a smaller stepsize worsens the exponentially decreasing factor in the convergence rate, but allows one to reach closer to the optimal value.

2.3 Polyak-Ruppert Averaging

SGD with larger stepsize converges faster during the transient phase albeit exhibits large oscillations when it reaches the asymptotic regime. In order to suppress the noise, it is may be useful to take appropriate averages of the sequence x_j that would automatically possess less noisy behavior. Polyak and Juditsky (1992) and Ruppert (1988) independently proposed a new optimal algorithm based on the idea of averaging the search point x_i:

$$\begin{aligned} x_t &= x_{t-1} - \eta_t \nabla f_t(x_{t-1}) \\ \bar{x}_t &= \frac{1}{t} \sum_{i=0}^{t-1} x_i. \end{aligned}$$

where the iterate averages $\bar{x}_t$ is returned as the estimate for the solution x_*. However, averaged sequence with a stepsize diminished with a rate of $O(1/t)$ performs poorly in practice. An important improvement in the use of iterate averaging came with the use of the longer stepsize diminishing in a slower rate:

$$\eta_t \to 0, \quad \frac{\eta_{t-1} - \eta_t}{\eta_{t-1}} = o(\eta_{t-1}).$$

The stepsize schedule $\eta_t = \eta t^{-\alpha}$ with $\alpha \in (0, 1)$ satisfies the restriction, but the sequence $\eta_t = \eta t^{-1}$ does not. An interesting result is that, with restricting $\alpha \in (\frac{1}{2}, 1)$, we can obtain optimal rate for the iterate averages $\mathbf{E}[\|\bar{x}_t - x_*\|^2] = O(1/t)$ while a worse rate for vanilla SGD $\mathbf{E}[\|x_t - x_*\|^2] = O(1/t^\alpha)$ [55, 61, 48]. In certain cases, this combination of long steps and averaging cancels out the adverse effects caused by ill-conditioning. In fact, SGD with averaging scheme can obtain the solution as good as full second order SGD [55]. While iterate average can take a very long time and a prohibitively large amount of data to reach its asymptotic regime with inappropriate stepsize sequence, a smart selection of the stepsize policy can significantly improve the performance [82].

On the other hand, the convergence rate of vanilla SGD for non-strongly convex objective is known to be $O(1/t^{1/3})$ [48], which does not match the optimal asymptotic minimax rate $O(1/\sqrt{t})$ for stochastic approximation in the non-strongly convex case [1, 49]. However, using long steps and averaging, one can show that the minimax asymptotic rate $O(1/\sqrt{t})$ can be obtained [50, 48].

2.4 Mini-Batching

Standard SGD suffers from the noisy gradients that become progressively less accurate when the iterate approaches the optimum. Instead of computing the gradient based one single example, one may use a mini-batch $\mathcal{I}_t$ of size b sampled i.i.d. from the data distribution:

$$x_t = x_{t-1} - \eta_t \frac{1}{b} \sum_{i_t \in \mathcal{I}_t} \nabla f_{i_t}(x_{t-1}).$$

It has been shown that one can reduce the variance [19]:

$$\mathbf{E}\left[\left\| \frac{1}{b} \sum_{i_t \in \mathcal{I}_t} \nabla f_{i_t}(x) - \nabla F(\theta) \right\|^2\right] \leq \frac{\sigma^2}{b}.$$

However, computing this mini-batch gradient is b times more expensive. Thus, there is a trade-off between "high variance with low per-iteration complexity" and "low variance with high per-iteration complexity".

Suppose that we employ mini-batch SGD with a sufficiently small constant stepsize η, convergence result (3) leads to

$$\mathbf{E}[F(x_t)] - F(x_*) \leq \frac{\eta L \sigma^2}{2b\mu} + (1 - \eta\mu)^t \left(F(x_0) - F(x_*) - \frac{\eta L \sigma^2}{2b\mu} \right).$$

Using standard SGD with a smaller step size η/b, we obtain a similar bound:

$$\mathbf{E}[F(x_t)] - F(x_*) \leq \frac{\eta L \sigma^2}{2b\mu} + \left(1 - \frac{\eta\mu}{b}\right)^t \left(F(x_0) - F(x_*) - \frac{\eta L \sigma^2}{2b\mu}\right).$$

The worse exponentially decreasing factor means that b times more iterations is needed for vanilla SGD to achieve the same suboptimality gap. Thus, two algorithms are comparable as they both require the same amount of total computational cost to obtain an equivalent suboptimality gap. However, the convergence result requires that the stepsize be smaller than $1/L$. Hence, one cannot simply adjust the stepsize for mini-batch SGD to be b times larger than the one used by vanilla SGD. In other words, this reduction of the variance does not necessarily pays for the higher per-iteration complexity.

However, in practice, mini-batching is still beneficial as it results in constant improvement in runtime (e.g. better throughput and efficient memory access) and offers opportunity for multi-core parallelization.

2.5 Stochastic vs. Batch Optimization Methods

While it is desirable to minimize (1) when the underlying distribution is the ground truth data distribution, such a goal is untenable unless we have acces to infinite number of fresh examples. Hence, in practice, one instead seeks to minimize an empirical risk (underlying distribution is the training set distribution in (1)) that can be expressed as the average of n loss functions:

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

In this case, batch gradient descent is a natural approach in which parameters are updated on the basis of the gradient evaluated on the entire training set:

$$x_t = x_{t-1} - \eta \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x),$$

where a constant stepsize η is used. For the minimization of the strongly convex objective F, the batch gradient method with a stepsize $\eta = 1/L$ enjoys a better linear convergence rate [51]:

$$F(x_t) - F(x_*) \leq \left(1 - \frac{\mu}{L}\right)^t [F(x_0) - F(x_*)].$$

Though the batch gradient method offers a better convergence result than SGD, its per-iteration complexity is n times larger than SGD. Thus, there is a trade-off in terms of per-iteration complexity and convergence rate in minimizing empirical risk.

To measure the total computational cost of the algorithms, let us define the ϵ-optimality as

$$\mathbf{E}[F(x)] - F(x_*) \leq \epsilon \text{ for some } \epsilon > 0.$$

Then, the total complexity required to obtain ϵ-optimality for the batch gradient method is $O(n\kappa \log(1/\epsilon))$, where $\kappa := L/\mu$ is the condition number. Thus, its total work grows linearly with the training set size n. (2) suggests that the total complexity required to obtain ϵ-optimality for the SGD is $O(\kappa/\epsilon)$, which is independent of the sample set size. Thus the comparison indicates that SGD is more favourable when n is large and one has limited computational time budget.

On the other hand, for machine learning community, what we care about is the generalization performance of the algorithm. The uniform laws of large numbers [76] suggests that the generalization error (difference between the expected and empirical risk) decreases at a rate of $O(\sqrt{\log n/n})$. Then, SGD is advantageous over the batch gradient method in that one can use as many examples as possible because its total complexity does not depend on n. However, for the batch gradient method, one have to increase ϵ in order to use more examples given a limited computational time budget. In particular, given a computational budget $\mathcal{T}_{\max}$, it can be shown that the batch gradient method obtains $\frac{\log \mathcal{T}_{\max}}{\mathcal{T}_{\max}} + \frac{1}{\mathcal{T}_{\max}}$-optimality for expected risk while the SGD can obtain $\frac{1}{\mathcal{T}_{\max}}$-optimality for expected risk under certain conditions [12].

3 Incremental Gradient Methods

The sound theoretical results together with extensive computational experience make SGD remain as one of the most notable and successful optimization method for large-scale applications. However, despite its scalability, the stochastic gradient is much noisier than the batch gradient. Thus, the stepsize has to be decreased gradually as stochastic learning proceeds, leading to slower convergence. In this section, I present ideas for incremental gradient methods in the context of minimizing a finite sum such as an empirical risk.

3.1 Finite Training Set

In many machine learning problems, one may consider the empirical risk minimization (ERM) problem:

$$\min_x F(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{4}$$

where n is the number of training samples, f_i is the loss due to sample i. We will assume that each f_i is Lipschitz smooth with constant L. We will also consider the case where each f_i is additionally strongly convex with constant μ. This setup differs from the traditional black box optimization problem only in that a finite sum structure is assumed. The common examples of ERM include linear regression problem and logistic regression problem.

With the finite sample size assumption, variance reduction [27] via control variates can used to reduce the variance of stochastic gradients in stochastic optimization. Specifically, given a random variable X and another highly correlated random variable Y, a variance-reduced estimate of $\mathbf{E}X$ can be obtained as

$$\tilde{X} = \alpha(X - Y) + \mathbf{E}Y, \tag{5}$$

where $\alpha \in [0, 1]$. When $\alpha = 1$, the estimate is unbiased $\mathbf{E}\tilde{X} = \mathbf{E}X$. The variance of $\text{Var}(\tilde{X}) = \alpha^2[\text{Var}(X) + \text{Var}(Y) - 2\text{Cov}(X, Y)]$, and so if X and Y is strongly positive correlated, the variance of $\tilde{X}$ can be reduced compared to X. By varying α from 1 to 0, we reduce the variance at the expense of increasing bias.

In stochastic optimization, the gradient $\nabla f_{i_t}(x_{t-1})$ with parameter x_{t-1} is taken as the X. By assuming that the training set is finite, various algorithms have been recently proposed so that Y is strongly correlated with $\nabla f_{i_t}(x_{t-1})$ while $\mathbf{E}Y$ can be efficiently evaluated. Examples include SAG [59], MISO [43], SVRG [36], Finito [18], SAGA [17], and SDCA [66]. With reduced variance in the stochastic gradients, a constant stepsize can be used, and the convergence rate of SGD can be significantly improved to match that of its batch counterpart. In the sequel, we will review a few notable examples of incremental gradient methods.

3.2 SAG

Algorithm 2 Stochastic Average Gradient (SAG).

Input: $\eta > 0$.
initialize $x_0 \in \mathbb{R}^d$; $\phi_i = \nabla f_i(x_0), \forall i \in [n]$; $d_0 = \sum_{i=1}^n \phi_i$;
for $t = 1, 2, \ldots$ **do**
  draw i_t uniformly at random in $\{1, \cdots, n\}$;
  $d_t = d_{t-1} - \phi_{i_t} + \nabla f_{i_t}(x_{t-1})$;
  $x_t = x_{t-1} - \frac{\eta}{n} d_t$;
  $\phi_{i_t} = \nabla f_{i_t}(x_{t-1})$;
end for

A prominent example along this direction is the method of stochastic average gradient (SAG) [59, 63], which reuses old stochastic gradients computed in previous iterations. It maintains a table of gradients of size $n \times d$ for all the components f_i. When f has a well structure (e.g., linear predictor), the storage requirement can be reduced to n [59, 36, 17]. The SAG method is shown in Algorithm 2. The key step is

$$d_t = d_{t-1} - \phi_{i_t} + \nabla f_{i_t}(x_{t-1}). \tag{6}$$

In fact, we move in a direction of the average of past gradients, where the past gradients are equally weighted. This is contrasted to the SGD method with momentum [75], which uses a geometric weighting of previous gradients. It is expected that (6) has less variance than the stochastic gradients employed in a basic SGD routine. Actually, SAG is an instance of (5) with $\alpha = 1/n$. Surprisingly, using such simple update can lead to much faster convergence speed compared with SGD, even though some of the stored gradients can be quite out-of-date and d_t is not an unbiased estimator of the full gradient $\nabla F(x_{t-1})$.

Beyond its initialization phase, the per-iteration complexity of Algorithm 2 is the same as in SGD. It has been shown that the method can achieve a linear rate of convergence for strongly convex problem:

$$\mathbf{E}[F(x_t)] - F(x_*) \leq \left(1 - \min\left(\frac{\mu}{16L}, \frac{1}{8n}\right)\right)^t C_0,$$

where C_0 is some constant. For non-strongly convex problem, a sublinear convergence rate can also be attained:

$$\mathbf{E}[F(\bar{x}_t)] - F(x_*) \leq \frac{32n}{t} C_0,$$

where $\bar{x}_t = \frac{1}{t}\sum_{k=0}^{t-1} x_k$. SAG algorithm is a randomized version of the incremental aggregated gradient (IAG) method proposed in [7] that uses a cyclic choice of i_t at each iteration. Interestingly, random selection yields faster convergence rate and better practical performance.

3.3 SAGA

Algorithm 3 SAGA.

Input: $\eta > 0$.
initialize $x_0 \in \mathbb{R}^d$; $\phi_i = \nabla f_i(x_0), \forall i \in [n]$;
for $t = 1, 2, \ldots$ **do**
  draw i_t uniformly at random in $\{1, \cdots, n\}$;
  $d_t = \nabla f_{i_t}(x_{t-1}) - \phi_{i_t} + \frac{1}{n}\sum_{i=1}^n \phi_i$;
  $x_t = x_{t-1} - \eta d_t$;
  $\phi_{i_t} = \nabla f_{i_t}(x_{t-1})$;
end for

The second algorithm we consider considers an iteration that is closer in form to SGD in that the gradient estimator is unbiased. Similar to SAG, it reuses the stochastic old gradients. The SAGA method is presented in Algorithm 3. The unbiased estimator used in SAGA is

$$d_t = \nabla f_{i_t}(x_{t-1}) - \phi_{i_t} + \frac{1}{n}\sum_{i=1}^n \phi_i, \tag{7}$$

where ϕ_{i_t} plays the role of the control variate Y and $\frac{1}{n}\sum_{i=1}^n \phi_i$ is its expectation $\mathbf{E}Y$. In SAG, the gradient estimator is biased away from the true gradient, whereas the gradient approximation (7) used in SAGA is unbiased. The trade-off for the increased bias is that SAG enjoys a smaller variance. In the following, we can rewrite SAG update for a clearer comparison:

$$\text{(SAG) } x_t = x_{t-1} - \eta\frac{1}{n}\left(\nabla f_{i_t}(x_{t-1}) - \phi_{i_t} + \sum_{i=1}^n \phi_i\right),$$

$$\text{(SAGA) } x_t = x_{t-1} - \eta\left(\nabla f_{i_t}(x_{t-1}) - \phi_{i_t} + \frac{1}{n}\sum_{i=1}^n \phi_i\right).$$

Note that the gradient difference term $\nabla f_{i_t}(x_{t-1}) - \phi_{i_t}$ is also weighted by $1/n$ for SAG. Such weighting scheme leads to a biased estimator but smaller variance as discussed in Section 3.1.

In the strongly convex case, when a step size of $\eta = 1/(2\mu n + L)$ is chosen, we have the following convergence rate for SAGA:

$$\mathbf{E}[\|x_t - x_*\|^2] \leq \left(1 - \frac{\mu}{2(\mu n + L)}\right)^t \left[\|x_0 - x_*\|^2 + \frac{n}{\mu n + L}[F(x_0) - F(x_*)]\right].$$

In the non-strongly convex case, the convergence rate is attained in terms of the average iterate. Let step size $\eta = 1/(3L)$, we have

$$\mathbf{E}[F(\bar{x}_t)] - F(x_*) \leq \frac{10n}{k}\left[\frac{2L}{n}\|x_0 - x_*\|^2 + F(x_0) - F(x_*)\right].$$

3.4 SDCA

Algorithm 4 Stochastic Dual Coordinate Descent (SDCA).

1: **initialize** $x_0 = \mathbf{0}$; $\alpha_0^i = \mathbf{0}, \forall i \in [n]$;
2: **for** $t = 1, 2, \ldots$ **do**
3: draw i_t uniformly at random in $\{1, \cdots, n\}$;
4: $\alpha_t^{i_t} = \alpha_{t-1}^{i_t} + \arg\max_{\triangle\alpha} \left\{-f_{i_t}^*(-\alpha_{t-1}^{i_t} - \triangle\alpha) - \frac{\lambda n}{2}\|x_{t-1} + \frac{1}{\lambda n}\triangle\alpha\|^2\right\}$;
5: $x_t = x_{t-1} - \frac{1}{\lambda n}(\alpha_{t-1}^{i_t} - \alpha_t^{i_t})$;
6: **end for**

The third method we review is the stochastic dual coordinate descent (SDCA) [66]. This method works on problem (4) with a quadratic regularizer, in which the dual enjoys a simple form to work with. Specifically, the SDCA algorithm considers following:

$$\min_x \frac{1}{n}\sum_{i=1}^n f_i(x) + \frac{\lambda}{2}\|x\|^2. \tag{8}$$

By applying Lagrangian duality theorem, the objective can be represented in the dual form:

$$\max_\alpha -\frac{1}{n}\sum_{i=1}^n f_i^*(-\alpha^i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n \alpha^i\right\|^2. \tag{9}$$

where f_i^* is the convex conjugate of f_i [1]. Essentially, SDCA randomly pick a dual variable α^{i_t} to update, and leave other dual variables unchanged. The SDCA algorithm is shown in Algorithm 4. As can be seen, SDCA is a randomized coordinate ascent method applied on dual objective. In step 5, we maintain x_t as the minimizer of the conjugate problem $\min_x \left[\frac{\lambda}{2}\|x\|^2 - \langle\frac{1}{n}\sum_{i=1}^n \alpha_t^i, x\rangle\right]$. The SDCA method has a linear convergence rate in the dual objective D of the form:

$$\mathbf{E}[D(\alpha_t)] - D(\alpha_*) \leq \left(1 - \frac{\lambda}{L + \lambda n}\right)^t [D(\alpha_t) - D(\alpha_*)].$$

[1] The convex conjugate of f is $f^*(y) = \sup_x y^T x - f(x)$

The goal of stochastic dual ascent is to increase the dual objective as much as possible in a randomized manner. However, it is not always easy to solve the maximization subproblem if f^* does not admit a closed-form solution. In general, we can apply nonlinear programming technique (e.g., Newton's method), but the extra computational cost can be expensive if we want to find an accurate solution. Instead, [66] proposed some alternative steps (e.g., 1-dimensional interval line search, constant step) to alleviate this problem while preserving the same theoretical convergence rate.

It has recently been shown that SDCA can work on primal problem (9) directly without using duality [65]. The SDCA without duality is shown in Algorithm 5. Observe that

Algorithm 5 Dual-Free SDCA.

Input: η.
initialize $\alpha_0^i = \mathbf{0}, \forall i \in [n]$; $x_0 = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_0^i$;
for $t = 1, 2, \ldots$ **do**
  draw i_t uniformly at random in $\{1, \cdots, n\}$;
  $\alpha_t^{i_t} = \alpha_{t-1}^{i_t} - \eta\lambda n(\nabla f_{i_t}(x_{t-1}) + \alpha_{t-1}^{i_t})$;
  $x_t = x_{t-1} - \eta(\nabla f_{i_t}(x_{t-1}) + \alpha_{t-1}^{i_t})$;
end for

SDCA maintains $x_t = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_t^i$. Conditioned on x_{t-1} and α_{t-1}, we have

$$\mathbf{E}[x_t] = x_{t-1} - \eta\left(\frac{1}{n}\sum_{i=1}^n \nabla f_i(x_{t-1}) + \lambda x_{t-1}\right)$$

That is, SDCA is in fact an instance of SGD. However, unlike the vanilla SGD, for SDCA the variance of the estimator goes to zero when the primal-dual parameters (x, α) converge to the optimal parameters (x_*, α_*). Specifically, it can be shown that

$$\mathbf{E}[\|\nabla f_{i_t}(x_{t-1}) + \alpha_{t-1}^{i_t}\|^2] \leq 2L\mathbf{E}[\|x_{t-1} - x_*\|^2 + 2\mathbf{E}\|\alpha_{t-1}^{i_t} - \alpha_*^{i_t}\|^2].$$

This indicates that the variance goes to zero asymptotically. The primal update used in Dual-Free SDCA allows us to deal with the case in which the objective is not explicitly regularized. In particular, we artificially add ℓ_2 regularizer to the objective and compensate for it by treating $f_{n+1}(x) = -\frac{\lambda(n+1)}{2}\|x\|^2$. The linear convergence rate still hold even though the individual loss functions are nonconvex, as long as the objective is strongly convex.

3.5 SVRG

While methods such as SAG, SAGA and SDCA require additional $O(nd)$ space, where n is the sample size and d is the model parameter dimensionality, the stochastic variance reduced gradient (SVRG) alleviates this storage problem by keeping snapshots of the full gradient and parameter estimator. Its extra memory requirement is low even when the sample size gets very large. This property makes SVRG more appealing for large-scale learning and neural network training [36, 2, 57]. SVRG is a multi-stage scheme (Algorithm 6). At

Algorithm 6 Stochastic Variance Reduced Gradient (SVRG).

Input: $\eta > 0$.
initialize $\tilde{x}_0 \in \mathbb{R}^d$;
for $s = 1, 2, \ldots, S$ **do**
  $x_0 = \tilde{x}_{s-1}$;
  $\tilde{z} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x}_{s-1})$;
  for $t = 1, 2, \ldots, m$ **do**
    draw i_t uniformly at random in $\{1, \cdots, n\}$;
    $d_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}_{s-1}) + \tilde{z}$;
    $x_t = x_{t-1} - \eta d_t$;
  end for
  Option 1: $\tilde{x}_s = x_m$;
  Option 2: $\tilde{x}_s = \frac{1}{m}\sum_{j=0}^{m-1} x_j$;
  Option 3: set $\tilde{x}_s = x_j$ for random j uniformly chosen from $\{0, \ldots, m-1\}$;
end for

the beginning of each stage, the full gradient $\tilde{z} = \frac{1}{n}\sum_{i=1}^{n} \nabla f(\tilde{x})$ is computed on the whole data set using an archived parameter estimate $\tilde{x}$ (which is updated across stages). For each subsequent iteration t in this stage, the following approximate gradient is used:

$$d_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{z},$$

By comparing with (5), it can be seen that $\nabla f_{i_t}(\tilde{x}_{s-1})$ plays the role of the control variate Y and $\tilde{z}$ is its expectation $\mathbf{E}Y$. When f is L-smooth, it can be shown that [2, 57]:

$$\mathbf{E}\left\| d_t - \frac{1}{n}\sum_{i=1}^{n} \nabla f(x_{t-1}) \right\|^2 \leq L^2 \|x_{t-1} - \tilde{x}\|^2, \tag{10}$$

Thus, the (expected) variance of d_t goes to zero when x_{t-1} and $\tilde{x}$ both converge to the same stationary point. However, this bound also indicates that the variance is bounded by the distance between the current iterate x_{t-1} and the archived estimate $\tilde{x}$. When x_{t-1} is far away from $\tilde{x}$, as during the early stage of optimization when x_{t-1} changes a lot, this variance can be large. Thus, SVRG can be sensitive to initialization, and is often initialized with SGD [36].

In order to reduce the variance in (10), $\tilde{x}$ needs to be updated frequently so that $\tilde{x}$ remains close to x. However, this also leads to more frequent full gradient computations so as to maintain $\tilde{z}$, and thus a high computation cost. In practice, $\tilde{x}$ is usually updated every $O(n)$ iterations. Thus, compared to SAG, SAGA and SAG, SVRG reduces the variance at the expense of high computational cost.

Theoretically, for both options 2 and 3, a linear convergence can be obtained for strongly convex problems:

$$\mathbf{E}[F(\tilde{x}_s)] - F(x_*) \leq \rho^s [F(\tilde{x}_0) - F(x_*)]$$

provided that appropriate η and m are chosen so that

$$\rho := \frac{1}{\eta\mu(1 - 2L\eta)m} + \frac{2L\eta}{1 - 2L\eta} < 1,$$

Very recently, motivated by the success of variance reduction in convex optimization, convergence results on nonconvex optimization have also been provided for SVRG [2, 57]. For nonconvex problems, it is hard to use the function value $\mathbf{E}\left[F(x_t) - F(x_*)\right]$ or distance between the final iterate and optimal solution $\mathbf{E}\|x_t - x_*\|^2$ as convergence criterion. Thus, it is more common to employ the notion of an ϵ-approximate stationary point x [26, 51], which means that $\|\nabla F(x)\|^2 \leq \epsilon$. It was shown that SVRG with option 1, $\eta = a/(Ln^{2/3})$ and $m = n/(3a)$ for some $0 < a < 1$, we can achieve

$$\mathbf{E}[\|\nabla F(x_o)\|^2] \leq \frac{Ln^{2/3}[F(\tilde{x}_0) - F(x_*)]}{Tv}$$

where x_o is chosen uniformly random from $\{\{x_t^s\}_{t=0}^{m-1}\}_{s=1}^{S}$, v is some universal constant, and $T = mS$. With variance reduction, the rate is improved to $O(1/T)$ in comparison to the slower $O(1/\sqrt{T})$ rate of SGD [26].

3.6 Summary

Although the incremental gradient methods achieve a faster convergence rate than SGD, their total complexity can suffer a linear dependency on the sample size. It has been shown that, for strongly convex problems, SGD requires $O(\kappa/\epsilon)$ gradient evaluations to find ϵ-optimality. It shows that the total work for SGD does not depend on the sample size. In comparison to SGD, stochastic variance reduction methods such as SAG, SAGA, SDCA and SVRG requires total complexity of the form:

$$O((n + \kappa)\log(1/\epsilon)), \tag{11}$$

which grows linearly with the training set size n. Thus, incremental gradient methods can be very expensive when the sample size is large. As indicated in [12], the incremental gradient methods can be as slow as batch gradient for very large n. However, the complexity bound (18) suggests that the incremental gradient methods may be superior when $\kappa \gg n$. At present, we should not regard the incremental gradient methods as clearly superior to SGD.

On the other hand, these variance reduction methods rely on the crucial assumption that the training set is finite, which is often valid in empirical loss minimization. However, it is usually more useful to minimize an expected loss over some data distribution [10, 11]. Variance reduction methods in stochastic optimization can no longer be used, as computing the full gradient then requires infinite samples. The finite training set assumption also fails in streaming data applications, where one does not have access to all the data samples. Another machine learning scenario is where data augmentation or random noise (such as dropout noise [68]) is added to the data for overfitting avoidance (as is common in neural network training). In this case, the resultant perturbed data set can be considered as infinite.

4 ADMM

Regularized risk minimization can be formulated as the following optimization problem

$$\min_x \frac{1}{n}\sum_{i=1}^{n} f_i(x) + r(x), \tag{12}$$

where x is the model parameter, n is the number of training samples, $f_i(x)$ is sample i's contribution to the empirical loss, and $r(x)$ is the regularizer (possibly non-smooth). If the non-differentiable term r is assumed to be simple so that its proximal operator is easy to evaluate, it is able to achieve the same convergence rate as if non-differentiable term is not included [53]. However, in many real cases, such as the (structured) sparsity regularizers, the non-differentiable term may be complex so that proximity operator does not have an analytical solution and is therefore difficult to compute. In this sense, some iterative methods are required to compute approximate proximity operator up to a certain accuracy. For example, the dual variant of Dykstra-like proximal splitting method has been used to compute approximate proximity operator for overlapping group lasso [84], and Newton-type methods that work with smooth dual reformulation was introduced to solve total variation based regularization [6]. For the learning of these complicated models, the alternating direction method of multipliers (ADMM), first introduced in [25], has been recently advocated as an efficient optimization solver [13].

For many (structured) sparsity regularizers, $r(x)$ is of the form $g(Ax)$, where A is a given linear transform. Examples include the lasso [73], total variation [60], graph-guided fused lasso [39] and overlapping group lasso [35]. By introducting an additional variable y, (12) can be equivalently written as

$$\min_{x,y} \; f(x) + g(y) \;:\; Ax - y = 0, \tag{13}$$

where

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{14}$$

In this section, I give a brief review on batch and stochastic ADMM algorithms that are particularly suitable for solving this type of problems.

4.1 Batch ADMM

The alternating direction method of multipliers (ADMM) [13] considers problems of the form

$$\min_{x,y} \; f(x) + g(y) \;:\; Ax + By = c, \tag{15}$$

where f, g are convex functions, and A, B (resp. c) are constant matrices (resp. vector) of appropriate sizes. The augmented Lagrangian of (15) is

$$L(x, y, \beta) \;\; = \;\; f(x) + g(y) + \beta^T(Ax + By - c) + \frac{\rho}{2}\|Ax + By - c\|^2,$$

where β is the dual variable and $\rho > 0$ is a penalty parameter. At the tth iteration, ADMM minimizes $L(x, y, \beta)$ w.r.t. x and y in an alternating manner. This allows the problem to be more easily decomposed when f and g are separable. Using the scaled dual variable $u = \beta/\rho$, the x and y updates can be expressed as:

$$\begin{aligned} x_t &= \arg\min_x f(x) + \frac{\rho}{2}\|Ax + By_{t-1} - c + u_{t-1}\|^2, \\ y_t &= \arg\min_y g(y) + \frac{\rho}{2}\|Ax_t + By - c + u_{t-1}\|^2, \end{aligned} \tag{16}$$

while u_t is updated as

$$u_t = u_{t-1} + Ax_t + By_t - c.$$

On convex problems, ADMM converges at a rate of $O(1/T)$, where T is the number of iterations [30]. On strongly convex problems, linear convergence can be attained [20, 34].

4.2 ADMM Using Stochastic Gradient

With $f(x)$ is given in (14), solving (16) can be computationally expensive when the data set is large. To alleviate this problem, a number of stochastic and online variants of ADMM have been developed very recently. Many of these are based on the standard SGD, and learn from only one sample (or a small mini-batch of samples) in each iteration. Examples include the online ADMM [77], stochastic ADMM (STOC-ADMM) [52], and online proximal gradient descent ADMM (OPG-ADMM) [71]. However, despite their low per-iteration complexities and good scalabilities, these algorithms only converge as $O(1/\sqrt{T})$ on convex problems and $O(\log T/T)$ on strongly convex problems. Hence, they are much slower than the batch ADMM, and can take many more iterations to converge. Specifically, OL-ADMM [77] replaces (16) with

$$x_t = \arg\min_x f_{i_t}(x) + \frac{\rho}{2}\|Ax + By_{t-1} - c + u_{t-1}\|^2 + \frac{D(x, x_{t-1})}{\eta_t},$$

where i_t is a sample randomly selected from $\{1, 2, \ldots, n\}$, $D(\cdot, \cdot)$ is a Bregman divergence (e.g., $\frac{1}{2}\|x - x'\|^2$), and $\eta_t \propto \frac{1}{\sqrt{t}}$. STOC-ADMM [52] changes the update rule for x_t to

$$x_t = \arg\min_x \nabla f_{i_t}(x_{t-1})^T x + \frac{\rho}{2}\|Ax + By_{t-1} - c + u_{t-1}\|^2 + \frac{\|x - x_{t-1}\|^2}{2\eta_t}.$$

It can be seen that the use of the linear approximation of f avoids a potentially expensive nonlinear programming step compared with OL-ADMM. However, the quadratic term $\|Ax\|^2$ also leads to an expensive inverse computation in each iteration, since x_t is given by

$$x_t = \left(\frac{1}{\eta_t} I + \rho A^T A\right)^{-1} \left(\frac{x_{t-1}}{\eta_t} - \nabla f_{i_t}(x_{t-1}) - \rho A^T (By_{t-1} - c + u_{t-1})\right).$$

Recently, Suzuki (2013) proposed a similar variant of ADMM called OPG-ADMM with $B = -I$ and $c = 0$, which further linearizes the quadratic term $\|Ax\|^2$ by replacing (16)

with

$$x_t \;=\; \arg\min_x \nabla f_{i_t}(x_{t-1})^T x + \frac{\rho}{2}\|Ax - y_{t-1} + u_{t-1}\|^2 + \frac{\|x - x_{t-1}|^2_{G_t}}{2\eta_t},$$

where $G_t = \gamma I - \eta_t \rho A^T A$ with a sufficiently large γ such that $G_t \succ 0$. This technique is known as inexact Uzawa method, which has been used in batch ADMM [87]. When G admits a general form under constraint that $G \succ 0$, it is also known as generalized ADMM [20]. Thus, OPG-ADMM updates x_t as

$$x_t \;=\; x_{t-1} - \frac{\eta_t}{\gamma}\left(\nabla f_{i_t}(x_{t-1}) + \rho A^T\left(Ax_{t-1} - \bar{y}_{t-1} + \bar{u}_{t-1}\right)\right).$$

As can be seen, the linearization of ADMM is advantageous over STOC-ADMM. Suzuki (2013) also proposed another variant called RDA-ADMM with $B = -I$ and $c = 0$ by utilizing the technique of regularized dual averaging [80]. RDA-ADMM maintains the averages of x, y, u and past gradients ∇f until t-th step as $\bar{x}_t = \frac{1}{t}\sum_{i=1}^t x_i, \bar{y}_t = \frac{1}{t}\sum_{i=1}^t y_i, \bar{u}_t = \frac{1}{t}\sum_{i=1}^t u_i$ and $\bar{g}_t = \frac{1}{t}\sum_{i=1}^t f_{i_t}(x_i)$, and changes (16) to

$$x_t \;=\; \arg\min_x \nabla \bar{g}_{t-1}^T x + \frac{\rho}{2t}\|Ax\|^2 + \rho(A\bar{x}_{t-1} - \bar{y}_{t-1} + \bar{u}_{t-1})^T Ax + \frac{\|x|^2_{G_t}}{2\eta_t},$$

where $\eta \propto \sqrt{t}$ and $G_t = \gamma I - \eta_t \rho A^T A/t$ with a sufficiently large γ such that $G_t \succ 0$. Then RDA-ADMM updates x_t as

$$x_t = -\frac{\eta_t}{\gamma}\left(\bar{g}_{t-1} + \rho A^T\left(A\bar{x}_{t-1} - \bar{y}_{t-1} + \bar{u}_{t-1}\right)\right).$$

4.3 Stochastic ADMM with Variance Reduction

Recently, variance reduction techniques developed for SGD have been incorporated into the stochastic variants of ADMM. In particular, Zhong and Kwok (2014) proposed the SAG-ADMM, Suzuki (2014) the SDCA-ADMM, and Zheng and Kwok (2016) the SVRG-ADMM. All enjoy low iteration complexities while achieving the same fast convergence as batch ADMM.

4.3.1 SAG-ADMM

Although the online algorithms proposed in [77, 52, 71] show low computational cost per iteration, it is a trade-off of fast computation and slow sublinear rate $O(1/T)$ to slow computation and fast linear rate. This means that stochastic method shows aggressive speed at first few epochs, but then gradually slows down, and is only effective if we only need an approximate solution. Recently, inspired by stochastic average gradient (SAG) [59], Zhong and Kwok (2014) proposed a variant of ADMM called SAG-ADMM, which incrementally updates the gradient as

$$\nabla \bar{f}_t = \frac{1}{n}\sum_{i=i}^n \nabla f_i(x_{\tau_i(t)}),$$

where

$$\tau_i(t) = \begin{cases} t-1 & i = i_t, \\ \tau_i(t-1) & \text{otherwise}, \end{cases}$$

The update rule for x_t of SAG-ADMM is

$$x_t = \frac{(\theta_1 \bar{x}_{t-1} + \theta_2 x_{t-1})}{\theta_1 + \theta_2} - \frac{1}{\theta_1 + \theta_2} \left(\nabla \bar{f}_t + \rho A^T (A x_{t-1} + B y_{t-1} - c + u_{t-1}) \right),$$

where $\theta_1 = \frac{1}{\eta_1}, \theta_2 = \frac{1}{\eta_2}$, and $\bar{x}_{t-1} = \frac{1}{n} \sum_{i=i}^{n} x_{\tau_i(t)}$. This method enjoys the low computational cost per iteration as stochastic ADMM whilst achieves $O(n/T)$ convergence rate in general (the rate for strongly convex is unknown). Recall that SAG needs to store n old gradients. Fortunately, in many machine learning models, it can be reduced to the storage of n scalars [36, 17]. However, SAG-ADMM still requires $O(nd)$ space even after using this trick, this comes from the need to store $\{x_{\tau_1(t)}, \ldots, x_{\tau_n(t)}\}$.

4.3.2 SDCA-ADMM

Very recently, motivated by SDCA, Suzuki (2014) develops SDCA-ADMM. Specifically, SDCA-ADMM considers the special case (13), where $B = -I$ and $c = 0$ as in (15). By applying Lagrangian duality theorem, the dual problem is obtained as

$$\begin{aligned} \min_{(\alpha,b)} \quad & \frac{1}{n} \sum_{i=1}^{n} f_i^*(\alpha^i) + g^* \left(\frac{b}{n} \right) \\ \text{s.t.} \quad & Z\alpha + A^T b = 0, \end{aligned}$$

where $Z = [z_1, \ldots, z_n] \in \mathbb{R}^{d \times n}$, f_i^* is convex conjugate of f_i, and g^* is convex conjugate of g. Then, a similar update strategy as stochastic ADMM can be applied on the dual. When A has full column rank, the rate of convergence of SDCA-ADMM is proven to be R-linear for strongly convex functions. Even though SDCA-ADMM shows exponential convergence, it still involves a nonlinear programming on the dual $f_{i_t}^*$ in each iteration. In that sense, it has the same disadvantage as OL-ADMM. As for the space requirement of SDCA-ADMM, it only needs $O(n)$ space for the dual variable, and may appear acceptable. However, the problem is aggravated when these algorithms are used in a large multi-class, multi-label or multitask learning problem. The space complexities will then be multiplied by a factor of N, where N is the number of label classes or tasks. Nowadays, it is not uncommon to have thousands or even millions of label classes. For example, Flickr has more than 20 millions tags, and the Dmoz data set has more than 30,000 labels. Hence, SAG-ADMM and SDCA-ADMM can be problematic when used on big data sets.

4.3.3 SVRG-ADMM

Motivated by SVRG [36], ADMM has recently been combined with SVRG in [89]. Unlike SAG-ADMM and SDCA-ADMM, SVRG-ADMM explicitly reduces the stochastic variance without having to store the past gradients or dual variables. This alleviates the space requirements mentioned above and can thus be even more scalable. Similiar to SVRG, for

SVRG-ADMM, the optimization procedure is divided into multiple stages, each with m iterations. At iteration t, the full gradient $\nabla f(x_{t-1})$ is approximated as

$$\hat{\nabla} f(x_{t-1}) = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{z}, \tag{17}$$

where $\tilde{z} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x})$. Note that $\hat{\nabla} f(x_{t-1})$ is unbiased. This approximate gradient is then used in the x_t update. The updates for x_t is therefore replaced with

$$x_t = \arg\min_x \left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{z}\right)^T x + \frac{\rho}{2}\|Ax + By_t - c + u_{t-1}\|^2 + \frac{\|x - x_{t-1}\|_G^2}{2\eta},$$

where G is some positive-definite matrix. By setting

$$G = \gamma I - \eta\rho A^T A \succeq I,$$

where

$$\gamma \geq \gamma_{\min} = \eta\rho\|A^T A\| + 1.$$

The x_t update (17) then simplifies to a gradient descent like update:

$$x_t = x_{t-1} - \frac{\eta}{\gamma}\left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{z} + \rho A^T(Ax_{t-1} + By_t - c + u_{t-1})\right).$$

It has been shown that SVRG-ADMM can obtain a linear convergence rate for strongly convex problems as well as a convergence rate of $O(1/T)$ for both non-strongly convex problems and nonconvex problems [88].

4.4 Summary

A summary of the space requirements and convergence rates for various stochastic ADMM algorithms is shown in Table 1. For simplicity, we consider $B = -I$ and $c = 0$. Moreover, we assume that the storage of the n old gradients can be reduced to the storage of n scalars, which is often the case in many machine learning models. As can be seen, SVRG-ADMM provides a more comprehensive convergence results than other stochastic ADMM methods. Among those with variance reduction, the space requirements of SVRG-ADMM is independent of the sample size n.

However, similar to the discussion in Section 3.6, the stochastic ADMM methods with variance reduction can not be viewed as clearly superior to other stochastic ADMM methods. In particular, the total computing times of SVRG-ADMM on strongly convex problems can be shown to be

$$O((n + \max(\kappa_f, \kappa_A\sqrt{\kappa_f}))\log(1/\epsilon)), \tag{18}$$

where κ_f is the condition number of f, and $\kappa_A = \sqrt{\sigma_{\max}(AA^T)/\sigma_{\min}(AA^T)}$, and $\sigma_{\max}(\cdot)$ (resp. $(\sigma_{\min}(\cdot))$) is its largest (resp. smallest) eigenvalue. Thus, the total complexity also grows linearly with the training set size n. At present, it is not known how useful these methods will prove to be in the future of large-scale machine learning.

Table 1: Convergence rates and space requirements of various stochastic ADMM algorithms, including stochastic ADMM (STOC-ADMM) [52], online proximal gradient descent ADMM (OPG-ADMM) [71], regularized dual averaging ADMM (RDA-ADMM) [71], stochastic averaged gradient ADMM (SAG-ADMM) [92], stochastic dual coordinate ascent ADMM (SDCA-ADMM) [72], and the stochastic variance reduced gradient ADMM (SVRG-ADMM) [88]. Here, $d, \tilde{d}$ are dimensionalities of x and y in (15).

	general convex	strongly convex	nonconvex	space requirement
STOC-ADMM	$O(1/\sqrt{T})$	$O(\log T/T)$	unknown	$O(d\tilde{d} + d^2)$
OPG-ADMM	$O(1/\sqrt{T})$	$O(\log T/T)$	unknown	$O(d\tilde{d})$
RDA-ADMM	$O(1/\sqrt{T})$	$O(\log T/T)$	unknown	$O(d\tilde{d})$
SAG-ADMM	$O(1/T)$	unknown	unknown	$O(d\tilde{d} + nd)$
SDCA-ADMM	unknown	linear rate	unknown	$O(d\tilde{d} + n)$
SVRG-ADMM	$O(1/T)$	linear rate	$O(1/T)$	$O(d\tilde{d})$

5 Optimization in Deep Learning

Recently, deep learning has emerged as a powerful and popular class of machine learning algorithms. Well-known examples include the convolutional neural network [42], long short term memory [33], memory network [78], and deep Q-network [47]. These models have achieved remarkable performance on various difficult tasks such as image classification [31], speech recognition [28], natural language understanding [5, 69], and game playing [67].

Deep network is a highly nonlinear model with typically millions of parameters [32]. Thus, it is imperative to design scalable and effective solvers. However, training deep networks is difficult as the optimization can suffer from pathological curvature and get stuck in local minima [44]. Moreover, every critical point that is not a global minimum is a saddle point [38], which can significantly slow down training. Second-order information is useful in that it reflects local curvature of the error surface. However, a direct computation of the Hessian is computationally infeasible. Martens (2010) introduced Hessian-free optimization, a variant of truncated-Newton methods that relies on using the linear conjugate gradient to avoid computing the Hessian. Natural gradient descent [3] uses the Fisher information matrix, and can produce a constant change on the functional manifold. Dauphin et al. (2014) proposed to use the absolute Hessian to escape from saddle points. However, these methods still require higher computational costs.

Recent advances in deep learning optimization focus mainly on SGD and its variants using the momentum techniques [70] as well as the algorithms with adaptive learning rates [22, 85, 74, 40, 90]. This section presents these optimization techniques for neural network training.

5.1 Basic Algorithms

We have previously introduced the SGD method in Section 2. This method has been viewed as the one of the most ideal optimization approaches for training deep learning models. Hardt et al. (2016) shows that SGD is uniformly stable, and therefore the solutions with low training error will achieve low generalization error as well. Though the incremental gradient methods reviewed in Section 3 enjoy faster local convergence, the fast initial progress made by SGD may matter a lot more than the asymptotic convergence for optimizing deep neural networks. In particular, in the context of learning from i.i.d. samples, $O(1/t)$ is asymptotically optimal due to the Cramér-Rao bound [14]. Therefore, it may not be necessary to pursue optimization algorithm that converges faster than $O(1/t)$ for training deep neural networks as deep models suffer a lot from overfitting. In the sequel, we will review momentum method, which can be used to further accelerate the training.

5.1.1 Momentum

The classic momentum method, also known as heavy ball method, introduced in [54], has been widely used in deep learning tasks. For a twice differentiable convex function, it is known that the momentum method can be used to accelerate the gradient descent and improve the convergence rate from $O((1-\kappa)^t)$ to $O(1-\sqrt{\kappa})^t$, where κ is the condition number of the functions. When training deep neural network, the momentum method is commonly combined with SGD for its low computational cost. It is known that momentum method has ability to accelerate early optimization and escape from the local minimums. The classic momentum method is:

$$x_t = x_{t-1} - \eta_t \nabla f_t(x_{t-1}) + \beta_t (x_{t-1} - x_{t-2}). \tag{19}$$

The iteration is equivalent to a discretization of the second-order ODE:

$$\ddot{x} + a\dot{x} + b\nabla f(x) = 0,$$

which models the motion of a body in a potential field given by f with friction, which motivates the initial naming of the algorithm by Polyak. It is common to set β_t to some constant β, e.g., 0.9. In this case, we can rewrite momentum method as

$$x_t = x_{t-1} - \sum_{i=1}^{t} \eta_i \beta^{t-i} \nabla f_i(x_{i-1}) \tag{20}$$

This reformulation shows that momentum can be seen as a technique that accumulates past directions of reduction in the objective across iterations, which serves as a smoothing of the velocity. When training deep neural networks, we usually consider a different formulation

$$\begin{aligned} m_t &= \beta m_{t-1} - \eta_t f_i(x_{t-1}), \\ x_t &= x_{t-1} + m_t. \end{aligned}$$

The complete momentum algorithm is presented in Algorithm 7.

Algorithm 7 SGD with Momentum.

Input: $\eta_t > 0$, $\beta \in [0, 1)$.
initialize $x_0 \in \mathbb{R}^d$; $m_0 = 0$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $m_t = \beta m_{t-1} - \eta_t \nabla f_t(x_{t-1})$;
  $x_t = x_{t-1} + m_t$;
end for

The main drawback of momentum method for stochastic optimization is that any advantages in terms of asymptotic local rate of convergence will be lost [79, 42]. However, the local fast convergence may not be very attractive in training deep neural networks. As discussed in [70], the local convergence in the final phase is not as important as fast training in the initial transient phase. In practice, we may stop training before entering the final steady state.

5.1.2 Nesterov Momentum

Algorithm 8 Nesterov Accelerated Gradient (NAG).

Input: $\eta_t > 0$, $\beta \in [0, 1)$.
initialize $x_0 \in \mathbb{R}^d$; $m_0 = 0$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $m_t = \beta m_{t-1} - \eta_t \nabla f_t(x_{t-1} + \beta m_{t-1})$;
  $x_t = x_{t-1} + m_t$;
end for

Sutskever et al. (2013) proposed to use Nesterov's accelerated gradient (NAG) [51] in training deep neural networks. NAG is closely related to the standard momentum method in that it can be written as:

$$\begin{aligned} m_t &= \beta m_{t-1} - \eta_t \nabla f_t(x_{t-1} + \alpha m_{t-1}), \\ x_t &= x_{t-1} + m_t. \end{aligned}$$

Nesterov momentum differs from the standard momentum in that the gradient is not evaluated w.r.t. to our current parameters but w.r.t. the approximate future position of our parameters. Thus one can interpret Nesterov momentum as attempting to add a correction factor to the vanilla momentum method. The Nesterov momentum algorithm is shown in Algorithm 8. NAG is argued to be more effective in the early or transient stage, and is more tolerant of large values of β compared to the vanilla momentum construction.

For general convex problems, Nesterov momentum improves the convergence rate from $O(1/t)$ to $O(1/t^2)$ [51]. Unfortunately, such benefit does not carry over to the stochastic gradient case. However, it has been shown that SGD with the diminishing Nesterov momentum will help accelerate the convergence rate during the initial transient phase when deterministic risk components dominate gradient noise [41].

5.1.3 Equivalence Results

Theoretically, the momentum construction can improve the convergence on deterministic optimization. However, such advantages may lose in stochastic setting. Recently, Yuan et al. (2016) shows that the SGD with (Nesterov) momentum is equivalent to the vanilla SGD for strongly convex functions under certain conditions. Specifically, if their stepsizes are small enough and satisfy following relation:

$$\eta_s = \frac{\eta_m}{1-\beta},$$

where η_s and η_m are the stepsizes for the vanilla SGD and the SGD with (Nesterov) momentum, respectively, then

$$\mathbf{E}[\|x_t^s - x_t^m\|^2] \leq O(\eta_s^{3/2}), \quad \forall t,$$

where x_t^s and x_t^m are the iterates at time t produced by the vanilla SGD and the SGD with (Nesterov) momentum, respectively. This result shows that the SGD and momentum methods evolve within $O(\eta_s^{3/2})$ from each other at all times. Therefore, the well-known acceleration of the momentum methods for deterministic optimization do not necessarily carry over to the stochastic optimization. This result also explains why the momentum method can achieve faster convergence during the transient phase as a larger stepsize for SGD does indeed lead to faster convergence but worse limiting performance [62]. However, such results only hold provided that the step size is sufficiently small for continuous adaptation. In practice, we may observe improvement using the momentum with a larger stepsize in some cases.

5.2 Adaptive Learning Rate in Deep Learning

The main bottleneck of SGD is that it requires careful stepsize tuning, which is difficult as different weights have vastly different gradients (in terms of both magnitude and direction). In particular, for a highly nonlinear convex model, the objective can be particularly more sensitive to some directions in parameter space. While momentum can alleviate this problem to some extent, it will be more useful if we can assign different stepsizes to different parameters. This section will briefly review a few of the adaptive learning rate algorithms.

5.2.1 Adagrad

Adaptive gradient descent (Adagrad) [22], presented in Algorithm 9, uses an adaptive per-coordinate stepsize:

$$\frac{\eta}{\sqrt{\sum_{i=1}^t g_i^2 + \epsilon}},$$

which is near-optimal in a certain sense [45]. Such adaptive learning rate scheme assigns smaller learning rates to the parameters with larger partial derivatives and larger learning rates to others with smaller partial derivatives. This can mitigate the exploding and

Algorithm 9 Adaptive Gradient Algorithm (Adagrad).

Input: $\eta > 0, \epsilon > 0$.
initialize $x_0 \in \mathbb{R}^d$; $s_0 = 0$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $g_t = \nabla f_t(x_{t-1})$;
  $s_t = s_{t-1} + g_t^2$;
  $x_t = x_{t-1} - \eta \frac{g_t}{\sqrt{s_t + \epsilon \mathbf{1}}}$;
end for

vanishing gradient problems somewhat. On convex problems, it has been shown both theoretically and empirically that Adagrad is especially efficient on high-dimensional data [22, 46]. In particular, its regret guarantee can be exponentially smaller in the dimension d than the non-adaptive regret bound. When used on deep networks, Adagrad also demonstrates significantly better performance than SGD [16]. However, in Adagrad, the variance estimate underlying the adaptive stepsize is based on accumulating all past (squared) gradients, which can result in a premature and excessive decrease in the effective learning rate. This becomes infinitesimally small as training proceeds.

5.2.2 RMSprop

Algorithm 10 RMSprop.

Input: $\eta_t > 0, \beta \in [0, 1), \epsilon > 0$.
initialize $x_0 \in x$; $v_0 = 0$;
for $t = 1, 2, \ldots, T$ **do**
  draw a function f_t;
  $g_t = \nabla f_t(x_{t-1})$;
  $v_t = \beta v_{t-1} + (1 - \beta) g_t^2$;
  $x_t = x_{t-1} - \eta_t \frac{g_t}{\sqrt{v_t + \epsilon \mathbf{1}}}$;
end for
Output: x_T.

A caveat as discussed in Section 5.2.1 is that Adagrad suffers from diminishing stepsize. As optimization proceeds, the accumulated squared gradient becomes larger and larger, making training difficult. RMSprop [74] alleviates the issue by estimating the variance using an exponentially decaying average of the squared gradients:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2. \tag{21}$$

Using a decay averaging discards the distant gradients that may not be very informative. The variance now instead becomes an approximation to local estimate. This ensures sufficient learning progress even after a large number of iterations of updates have been done.

Empirically, RMSProp has been shown to be an effective optimization algorithm for online and non-stationary settings. It has been viewed as one of the most popular methods for the training of deep learning models by practitioners.

Algorithm 11 Adaptive Moment Estimation (Adam).

Input: $\eta_t > 0$, $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$.
initialize $x_0 \in \mathbb{R}^d$; $m_0 = 0$; $v_0 = 0$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $g_t = \nabla f_t(x_{t-1})$;
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$;
  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$;
  $\tilde{m}_t = \frac{v_t}{1-\beta_2^t}$;
  $\tilde{v}_t = \frac{m_t}{1-\beta_1^t}$;
  $x_t = x_{t-1} - \eta_t \frac{\tilde{m}_t}{\sqrt{\tilde{v}_t + \epsilon \mathbf{1}}}$;
end for

5.2.3 Adam

Note that the variance estimate (21) used in RMSprop can be rewritten as

$$v_t = \sum_{i=1}^{t} (1 - \beta) \beta^{t-i} g_i^2, \tag{22}$$

as v_0 is initialized to 0. The estimate becomes zero when $\beta \to 1$, and therefore blows up the stepsize, leading to divergence. Thus, the bias has to be corrected. Adam [40] uses the variance estimate $v_t/(1 - \beta^t)$, which can be shown to be able to recover the global estimate used in Adagrad:

$$\lim_{\beta \to 1} \frac{v_t}{1 - \beta^t} = \frac{1}{t} \sum_{i=1}^{t} g_i^2.$$

In practice, a smaller β has to be used for RMSprop. However, a larger β enables the algorithm to be more robust to the gradient noise in general.

In Section 5.1.1, we have shown that momentum method can help accelerate the training. Adam also incorporates momentum via adding a direct moving averaging of the gradients to RMSprop though there is no a clear theoretical justification on such combination with adaptive learning rate as mentioned in [24]. The complete Adam method is presented in Algorithm 11. Adam is observed to be more robust to the hyperparameters than RMSprop in practical experience [24].

5.2.4 FTML

Online learning [93], which is closely related to stochastic optimization, has been extensively studied in the past decade. Well-known algorithms include follow the regularized leader (FTRL) [37] and follow the proximally-regularized leader (FTPRL) [45]. Motivated by these algorithms, follow the moving leader (FTML) [90] is yet another algorithm presented in Algorithm 12 that considers solving following subproblem at round t:

$$x_t = \arg\min_x \sum_{i=1}^{t} w_{i,t} \left(\langle g_i, x \rangle + \frac{1}{2} \|x - x_{i-1}\|^2_{\text{diag}\left(\frac{\sigma_i}{1-\beta_1}\right)} \right). \tag{23}$$

Algorithm 12 Follow the Moving Leader (FTML).

Input: $\eta_t > 0$, $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$.
initialize $x_0 \in \mathbb{R}^d$; $d_0 = 0$; $v_0 = 0$; $z_0 = 0$;
for $t = 1, 2, \ldots$ **do**
  draw a function f_t;
  $g_t = \nabla f_t(x_{t-1})$;
  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$;
  $d_t = \frac{1-\beta_1^t}{\eta_t} \left(\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon \mathbf{1} \right)$;
  $\sigma_t = d_t - \beta_1 d_{t-1}$;
  $z_t = \beta_1 z_{t-1} + (1 - \beta_1) g_t - \sigma_t x_{t-1}$;
  $x_t = -z_t / d_t$;
end for

where the weights

$$w_{i,t} = \frac{(1 - \beta_1)\beta_1^{t-i}}{1 - \beta_1^t}$$

are normalized to sum to 1. In (23), the recent samples are weighted more heavily in each iteration and so FTML can adapt more quickly to changes. The denominator $1 - \beta_1^t$ plays a similar role as bias correction in Adam. σ_i's are set so that following relation holds:

$$\sum_{i=1}^{t} w_{i,t} \frac{\sigma_i}{1 - \beta_1} = \text{diag}\left(\frac{1}{\eta_t} \left(\sqrt{\frac{v_t}{1 - \beta_2^t}} + \epsilon_t \mathbf{1} \right) \right),$$

which is also used in Adam for the variance estimation. Note that some entries of σ_i may be negative, and $\|x - x_{i-1}\|^2_{\text{diag}(\sigma_i/(1-\beta_1))}$ is then not a regularizer in the usual sense. Instead, the negative entries of σ_i encourage the corresponding entries of x to move away from those of x_{i-1}. Nevertheless, the objective in (23) is still strongly convex.

FTML has a strong connection with Adam. At iteration t, instead of centering regularization at each x_{i-1} in (23), consider centering all the proximal regularization terms at the last iterate x_{t-1}. x_t then becomes:

$$\arg\min_x \sum_{i=1}^{t} w_{i,t} \left(\langle g_i, x \rangle + \frac{1}{2} \|x - x_{t-1}\|^2_{\text{diag}\left(\frac{\sigma_i}{1-\beta_1}\right)} \right). \tag{24}$$

Compared with (23), the regularization in (24) is more aggressive as it encourages x_t to be close only to the last iterate x_{t-1}.

Note that the x_t updates of Adagrad, RMSprop, and FTML depend only on the current gradient g_t. On the other hand, the Adam update involves momentum, which is sometimes helpful in escaping from local minimum. However, when the data distribution is changing (as in deep reinforcement learning), the past gradients may not be very informative, and can even impede parameter adaptation to the environment. Recently, it is also reported that the use of momentum can make training unstable when the loss is nonstationary [4].

Thus, FTML enjoys the nice properties of RMSprop and Adam, while avoiding their pitfalls. It has been shown that FTML shows good stable performance on a number of deep learning models and tasks.

6 Conclusion and Future Research

In this survey, we highlight the prominent SGD method, whose success motivates a class of optimization afterwards. Though SGD offers promising scalability, it suffers from the slow convergence. Thus, it may not be the best suited method for the emerging computer architecture. This leads to the discussion on incremental gradient methods that have been demonstrated to obtain better convergence rates and have the potential to surpass SGD in the next generation of optimization methods for machine learning. we also discuss a spectrum of stochastic ADMM methods that offer ability to solve large-scale problems with complex structure. Last but not least, we turn the attention to the optimization methods applied on deep learning. Deep learning has been identified a current trend in machine learning community. However, training deep networks is difficult as the optimization can suffer from pathological curvature and get stuck in local minima. Thus, it is imperative to design scalable and effective solvers. Recent advances in deep learning optimization ranges from SGD and its variants with momentum to the adaptive learning rate algorithms. In the future, one can expect the opportunities offered by parallel and distributed computing as well as a more scalable incremental gradient method that can be applied to more scenario other than the minimization of the empirical risk.

References

[1] A. Agarwal, M. J. Wainwright, P. L. Bartlett, and P. K. Ravikumar. Information-theoretic lower bounds on the oracle complexity of convex optimization. In *Advances in Neural Information Processing Systems*, pages 1–9, 2009.

[2] Z. Allen-Zhu and E. Hazan. Variance reduction for faster non-convex optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[3] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2): 251–276, 1998.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. Preprint arXiv:1701.07875, 2017.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference for Learning Representations*, 2015.

[6] A. Barbero and S. Sra. Fast newton-type methods for total variation regularization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 313–320, 2011.

[7] D. Blatt, A. O. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[8] L. Bottou. Stochastic learning. In *Advanced Lectures on Machine Learning*, pages 146–168. Springer Verlag, 2004.

[9] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*, page 177, 2010.

[10] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168, 2008.

[11] L. Bottou and Y. Lecun. Large scale online learning. In *Advances in Neural Information Processing Systems*, pages 217–224, 2004.

[12] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. Preprint arXiv:1606.04838, 2016.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[14] G. Casella and R. L. Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[15] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

[16] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, and A. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[17] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 2116–2124, 2014.

[18] A. Defazio, J. Domke, and T. Caetano. Finito: A faster, permutable incremental gradient method for big data problems. In *Proceedings of the 31st International Conference on Machine Learning*, page 11251133, 2014.

[19] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1):165–202, 2012.

[20] W. Deng and W. Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, pages 1–28, 2015.

[21] J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

[22] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[23] R. Frostig, R. Ge, S. M. Kakade, and A. Sidford. Competing with the empirical risk minimizer in a single pass. In *Proceedings of the 28th Conference on Learning Theory*, pages 728–763, 2015.

[24] Ian G., Yoshua B., and Aaron C. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[26] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[27] P. Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53. Springer, 2013.

[28] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[29] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1225–1234, 2016.

[30] B. He and X. Yuan. On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[32] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[33] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[34] M. Hong and Z.-Q. Luo. On the linear convergence of the alternating direction method of multipliers. Technical Report arXiv:1208.3922, University of Minnesota, 2012.

[35] L. Jacob, G. Obozinski, and J.-P. Vert. Group lasso with overlap and graph lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 433–440, 2009.

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

[37] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[38] K. Kawaguchi. Deep learning without poor local minima. In *Advances In Neural Information Processing Systems*, pages 586–594, 2016.

[39] S. Kim, K. A. Sohn, and E. P. Xing. A multivariate regression approach to association analysis of a quantitative trait network. *Bioinformatics*, 25(12):i204–i212, 2009.

[40] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations*, 2015.

[41] G. Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1):365–397, 2012.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[43] J. Mairal. Optimization with first-order surrogate functions. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[44] J. Martens. Deep learning via Hessian-free optimization. In *Proceedings of the International Conference on Machine Learning*, pages 735–742, 2010.

[45] H. B. McMahan and M. Streeter. Adaptive bound optimization for online convex optimization. *In Proceedings of the Twenty Third Annual Conference on Computational Learning Theory*, page 244, 2010.

[46] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 1222–1230, 2013.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

[48] E. Moulines and F. R. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

[49] A. Nemirovski and D.B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.

[50] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[51] Y. Nesterov. *Introductory lectures on convex optimization*, volume 87. Springer Science & Business Media, 2004.

[52] H. Ouyang, N. He, L. Tran, and A. Gray. Stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning*, pages 80–88, 2013.

[53] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[54] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

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

[56] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

[57] S. J. Reddi, A. Hefny, S. Sra, B. Póczós, and A. Smola. Stochastic variance reduction for nonconvex optimization. In *Proceedings of the 33rd international conference on Machine learning*, 2016.

[58] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[59] N.L. Roux, M. Schmidt, and F.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.

[60] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[61] D. Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[62] A. H. Sayed. Adaptation, learning, and optimization over networks. *Foundations and Trends® in Machine Learning*, 7(4-5):311–801, 2014.

[63] M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[64] S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[65] S. Shalev-Shwartz. SDCA without duality, regularization, and individual convexity. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 747–754, 2016.

[66] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

[67] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, George Van D. D., J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[68] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[69] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2440–2448, 2015.

[70] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

[71] T. Suzuki. Dual averaging and proximal gradient descent for online alternating direction multiplier method. In *Proceedings of the 30th International Conference on Machine Learning*, pages 392–400, 2013.

[72] T. Suzuki. Stochastic dual coordinate ascent with alternating direction method of multipliers. In *Proceedings of the 31st International Conference on Machine Learning*, pages 736–744, 2014.

[73] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, pages 267–288, 1996.

[74] T. Tieleman and G. Hinton. Lecture 6.5 - RMSProp, COURSERA: Neural networks for machine learning, 2012.

[75] P. Tseng. An incremental gradient (-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, 8(2):506–531, 1998.

[76] V. N. Vapnik. *Estimation of dependences based on empirical data*, volume 40. Springer-Verlag New York.

[77] H. Wang and A. Banerjee. Online alternating direction method. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1119–1126, 2012.

[78] J. Weston, S. Chopra, and A. Bordes. Memory networks. Preprint arXiv:1410.3916, 2014.

[79] W. Wiegerinck, A. Komoda, and T. Heskes. Stochastic dynamics of learning with momentum in neural networks. *Journal of Physics A: Mathematical and General*, 27 (13):4425, 1994.

[80] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596, 2010.

[81] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4), 2014.

[82] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. Preprint arXiv:1107.2490, 2011.

[83] K. Yuan, B. Ying, and A. H. Sayed. On the influence of momentum acceleration on online learning. *Journal of Machine Learning Research*, 17(192):1–66, 2016.

[84] L. Yuan, J. Liu, and J. Ye. Efficient methods for overlapping group lasso. In *Advances in Neural Information Processing Systems*, pages 352–360, 2011.

[85] M. D. Zeiler. ADADELTA: An adaptive learning rate method. Preprint arXiv:1212.5701, 2012.

[86] R. Zhang, S. Zheng, and J. T. Kwok. Asynchronous distributed semi-stochastic gradient optimization. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2323–2329, 2016.

[87] X. Zhang, M. Burger, and S. Osher. A unified primal-dual algorithm framework based on Bregman iteration. *Journal of Scientific Computing*, 46(1):20–46, 2011.

[88] S. Zheng and J. T. Kwok. Stochastic variance-reduced admm. Preprint arXiv:1604.07070, 2016.

[89] S. Zheng and J.T. Kwok. Fast-and-light stochastic admm. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 2407–2413, 2016.

[90] S. Zheng and J.T. Kwok. Follow the moving leader in deep learning. In *Proceedings of The 34rd International Conference on Machine Learning*, pages 4110–4119, 2017.

[91] S. Zheng, R. Zhang, and J. T. Kwok. Fast nonsmooth regularized risk minimization with continuation. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2393–2399, 2016.

[92] W. Zhong and J. Kwok. Fast stochastic alternating direction method of multipliers. In *Proceedings of the 31st International Conference on Machine Learning*, pages 46–54, 2014.

[93] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. 2003.