Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Parallel Asynchronous Stochastic Variance Reduction for Nonconvex Optimization

Cong Fang, Zhouchen Lin*
Key Laboratory of Machine Perception (MOE), School of EECS, Peking University, P. R. China
Cooperative Medianet Innovation Center, Shanghai Jiao Tong University, P. R. China
fangcong@pku.edu.cn, zlin@pku.edu.cn

Abstract

Nowadays, asynchronous parallel algorithms have received much attention in the optimization field due to the crucial demands for modern large-scale optimization problems. However, most asynchronous algorithms focus on convex problems. Analysis on nonconvex problems is lacking. For the Asynchronous Stochastic Descent (ASGD) algorithm, the best result from (Lian et al. 2015) can only achieve an asymptotic $O(\frac{1}{\epsilon^2})$ rate (convergence to the stationary points, namely, $\|\nabla f(\mathbf{x})\|^2 \leq \epsilon$) on nonconvex problems. In this paper, we study Stochastic Variance Reduced Gradient (SVRG) in the asynchronous setting. We propose the Asynchronous Stochastic Variance Reduced Gradient (ASVRG) algorithm for nonconvex finite-sum problems. We develop two schemes for ASVRG, depending on whether the parameters are updated as an atom or not. We prove that both of the two schemes can achieve linear speed up[1](a non-asymptotic $O(\frac{n^{\frac{2}{3}}}{\epsilon})$ rate to the stationary points) for nonconvex problems when the delay parameter $\tau \leq n^{\frac{1}{3}}$, where n is the number of training samples. We also establish a non-asymptotic $O(\frac{n^{\frac{2}{3}}\tau^{\frac{1}{3}}}{\epsilon})$ rate (convergence to the stationary points) for our algorithm without assumptions on τ. This further demonstrates that even with asynchronous updating, SVRG has less number of Incremental First-order Oracles (IFOs) compared with Stochastic Gradient Descent and Gradient Descent. We also conduct experiments on a shared memory multi-core system to demonstrate the efficiency of our algorithm.

Introduction

We study nonconvex finite-sum problems of the form:

$$\min_{\mathbf{x}\in\mathcal{R}^d} f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}), \tag{1}$$

where f_i ($i \in \{1, 2, \cdots, n\}$) have L-Lipschitz continuous gradient ($L > 0$) but can be nonconvex and n is the number of functions. A large number of models can be formulated as Eq. (1), such as neural networks, dictionary learning, and inference in graphical models (Allen-Zhu and Hazan 2016). In this paper, we focus on algorithms that can efficiently reach a stationary point satisfying $\|\nabla f(\mathbf{x})\|^2 \leq \epsilon$, which is a common benchmark for nonconvex algorithms.

The standard method to solve Eq. (1) is through Gradient Descent (GD) and Stochastic Gradient Descent (SGD). In large-scale problems, SGD is faster in practice, since it randomly chooses only one sample to estimate the gradient during each update. However, its provable convergence rate is slower than GD. The Incremental First-order Oracles (IFOs) complexity for GD and SGD to reach a stationary point are $O(\frac{n}{\epsilon})$ and $O(\frac{\sigma}{\epsilon^2})$, respectively, where σ is the variance of stochastic gradient (Ghadimi and Lan 2013).

Variance Reduction (VR) methods are one of the great varieties of SGD methods which ensure the descent direction to have a bounded variance and so can achieve a much faster convergence rate compared with SGD. (Johnson and Zhang 2013) first propose the Stochastic Variance Reduced Gradient (SVRG) algorithm and prove that the algorithm has a linear convergence rate instead of a sublinear rate for SGD, for strongly convex problems. They have also done a compelling experiment on neural networks to demonstrate the advantage on nonconvex problems. Recently, there are much research (Reddi et al. 2016), (Allen-Zhu and Hazan 2016) that carefully analyse SVRG on nonconvex optimization problems. They both prove that SVRG convergences in $O(\frac{n^{\frac{2}{3}}}{\epsilon})$ for nonconvex problems, which is at least $O(n^{\frac{1}{3}})$ faster than GD.

On the other hand, to meet the requirement for modern large-scale problems, asynchronous parallel algorithms have received much attention, e.g., Asynchronous Stochastic Gradient Descent (ASGD) (Niu et al. 2011),(Agarwal and Duchi 2011), (Lian et al. 2015), Asynchronous Coordinate Descent (Liu et al. 2015), Asynchronous Dual Coordinate Descent (Hsieh, Yu, and Dhillon 2015), and Asynchronous Alternating Direction Method of Multipliers (Zhang and Kwok 2014). Most existing asynchronous methods focus on convex optimization. So the analysis on large-scale nonconvex problems, such as neural networks, is lacking. On nonconvex problems, the ASGD algorithm (Lian et al. 2015) can

*Corresponding author.

[1]The linear speed up means that if we use τ cores to solve a problem, it will be at least $\alpha\tau$ times faster than using only one core to solve this problem ($\alpha > 0$). It indicates that the asynchronous algorithm can still maintain the same convergence rate with the serial algorithm when we ignore the constant in the convergence rate.

only achieve an asymptotic $O(\frac{1}{\epsilon^2})$ rate. The convergence rates for different algorithms are shown in Table 1.

In this paper, we study the asynchronous variant of SVRG for nonconvex optimization problems. We choose SVRG rather than other VR methods since it has a low storage requirement, which is more suitable for optimization with a large number of variables, such as neural networks. We develop two schemes for Asynchronous Stochastic Variance Reduced Gradient (ASVRG) depending on whether the parameters are updated as an atom. Distinguished from the proof of ASGD (Lian et al. 2015), we propose a unified proof for the two schemes. We show that both schemes can achieve a linear speed up (a non-asymptotic $O(\frac{n^{\frac{2}{3}}}{\epsilon})$ rate to stationary points) for nonconvex problems when the delay parameter $\tau \leq n^{\frac{1}{3}}$. We also establish a non-asymptotic $O(\frac{n^{\frac{2}{3}}\tau^{\frac{1}{3}}}{\epsilon})$ rate (convergence to the stationary point) without any assumption on τ. This demonstrates that with asynchronous updating, SVRG still has a less number of IFOs compared with SGD and GD. We then experiment on a shared memory system to validate the speedup properties and demonstrate the efficiency of our algorithm. In summary, our work makes the following contributions:

- We devise the ASVRG algorithm for asynchronous large-scale *nonconvex* optimization or distributed systems.
- We show that ASVRG can achieve linear speed up when $\tau \leq n^{\frac{1}{3}}$ and also prove that it has an ergodic $O(\frac{n^{\frac{2}{3}}\tau^{\frac{1}{3}}}{\epsilon})$ convergence rate with no assumption on τ.

Related Work

Asynchronous Parallel Algorithms

Asynchronous algorithms have achieved great success in recent years. Up to now, there are lots of practical algorithms. Due to space limit, we only review the algorithms that have close relation with ours. The first work is from (Niu et al. 2011), which proposes a lock free asynchronous implementation of SGD on a shared memory system, called HOGWILD!. They also provide a proof of non-asymptotic $O(1/\epsilon)$ convergence rate for strongly convex and smooth objective functions. (Agarwal and Duchi 2011) propose an implementation of SGD on computer clusters for convex smooth problems and prove that the convergence rate is $O(1/\epsilon^2 + \tau^2/\epsilon)$. (Liu et al. 2015) propose an asynchronous stochastic coordinate descent algorithm and prove that the near-linear speed up is achievable if $\tau \leq O(n^{1/2})$ for smooth convex functions under certain conditions. Another two works we should mention are (Reddi et al. 2015) and (Lian et al. 2015). In the first work, (Reddi et al. 2015) study asynchronous SVRG on strongly convex functions. They achieve a linear convergence rate under certain conditions. For nonconvex problem, (Lian et al. 2015) analyse ASGD and show that ASGD can achieve an asymptotic $O(1/\epsilon^2)$ convergence rate.

Table 1: Convergence rates of GD based algorithms to stationary points of non-convex problems ("Syn." indicates the algorithm is serial or synchronous in the minibatch mode, while "asyn." indicates that the algorithm is asynchronous. Asym. is short for asymptotic. τ is the delay parameter.).

	Algorithm	Convergence Rate
Syn.	GD (Nesterov 2013)	non-asym. $O(\frac{n}{\epsilon})$
	SGD (Ghadimi and Lan 2013)	non-asym. $O(\frac{1}{\epsilon^2})$
	SVRG (Reddi et al. 2016)	non-asym. $O(\frac{n^{\frac{2}{3}}}{\epsilon})$
Asyn.	ASGD (Lian et al. 2015)	asym. $O(\frac{1}{\epsilon^2})$
	ASVRG (ours)	non-asym. $O(\frac{n^{\frac{2}{3}}}{\epsilon})$, $\tau \leq n^{\frac{1}{3}}$
		non-asym. $O(\frac{n^{\frac{2}{3}}\tau^{\frac{1}{3}}}{\epsilon})$

Algorithm 1 Serial SVRG

Input $\mathbf{x}_0^0$, epoch length m, step size γ, and $S = \lfloor K/m \rfloor$.
for $s = 0$ **to** $S-1$ **do**
  $\mathbf{g}^s = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\mathbf{x}_0^s)$,
  for $k = 0$ **to** $m-1$ **do**
    Randomly sample i_k from $1, 2, \cdots, n$,
    $\mathbf{v}_k^s = \nabla f_{i_k}(\mathbf{x}_k^s) - \nabla f_{i_k}(\mathbf{x}_0^s) + \mathbf{g}^s$,
    $\mathbf{x}_{k+1}^s = \mathbf{x}_k^s - \gamma \mathbf{v}_k^s$,
  end for k.
  $\mathbf{x}_0^{s+1} = \mathbf{x}_m^s$,
end for s.

VR Methods

VR methods have received a broad attention in recent years, e.g., SAG (Schmidt, Roux, and Bach 2013), SVRG (Johnson and Zhang 2013), and SAGA (Defazio, Bach, and Lacoste-Julien 2014).

Since we focus on the asynchronous variant of SVRG, we review SVRG in detail. The algorithm has double loops. In the outer loop, SVRG defines a snapshot vector $\mathbf{x}_0^s$, and computes the full gradient. At the inner loop, SVRG computes the following gradient estimator:

$$\mathbf{v}_k^s = \nabla f_{i_k}(\mathbf{x}_k^s) - \nabla f_{i_k}(\mathbf{x}_0^s) + \nabla f(\mathbf{x}_0^s). \quad (2)$$

The algorithm is shown in Algorithm 1, where γ is step size and m is the epoch length. To explain that $\mathbf{v}_k^s$ has been reduced variance, one can consider the case when $\mathbf{x}_k^s$ is very close to a stationary point $\mathbf{x}^*$. Then we have

$$\mathbf{v}_k^s \approx \nabla f_{i_k}(\mathbf{x}^*) - \nabla f_{i_k}(\mathbf{x}^*) + \nabla f(\mathbf{x}^*) = 0, \quad (3)$$

while for SGD, the gradient estimator is $\nabla f_{i_t}(\mathbf{x}^*)$, which is not equal to 0. Another observation from Eq. (3) is that the step size need not decrease to 0 to ensure the convergence, which is different from SGD. In (Johnson and Zhang 2013), the authors prove that it can achieve a linear convergence rate for strongly convex problems. Then (Xiao and Zhang 2014) study the algorithm in the general convex case. Recently, in (Reddi et al. 2016) and (Allen-Zhu and Hazan 2016), the authors analyse the algorithm in general nonconvex problems. They show that SVRG convergences in $O(\frac{n^{\frac{2}{3}}}{\epsilon})$. They bound the variance of gradient through the following equation

$$\mathbb{E}\left(\|\mathbf{v}_k^s\|^2\right) \leq \mathbb{E}\left(\|\nabla f(\mathbf{x}_k^s)\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}_k^s - \mathbf{x}_0^s\|^2\right). \quad (4)$$

Our Algorithms

Due to the rapid development of hardware resources, asynchronous parallelisms have recently been very successful on many problems, including nonconvex ones, such as neural networks (Dean et al. 2012),(Paine et al. 2013) and matrix decomposition (Petroni and Querzoni 2014),(Yun et al. 2014). The advantage of asynchronous parallelisms is that it allows workers to work independently. So it reduces the system overhead. Though SVRG has a provably faster convergence rate on non-convex problems using one core, the asynchronous variant of SVRG, which meets the crucial demands for large-scale optimization, has not been studied. We study two schemes of ASVRG. The first scheme ensures the parameter to be updated as an atom, which is common in star-shaped computer networks, while the other scheme has no locks during the updates. This scheme is common in shared memory multi-core systems.

ASVRG-atom

We first describe ASVRG-atom, which is common in star-shaped computer networks. Since SVRG has two loops, we implement SVRG in asynchronization in the inner loop. In each epoch, we compute $\nabla f(\mathbf{x}_0^s)$ and update $\mathbf{x}_k^s$ in asynchronization. There will be a synchronization operation after computing the full gradient and after each epoch, respectively. Since both m and n are always large, the synchronization operation will not cost much time.

More specifically, we first assign 2 global counters k and j. Then all threads repeat the following two parts independently and simultaneously:

Part I: Computing the Full Gradient

1) **(Read)** Read the parameter $\tilde{\mathbf{x}}_0^s$ from the global memory to the local memory without locks, and set $j = 0$.
2) **(Loop)** While $j < n$
3) $j = j + 1$, globally,
4) Compute the gradient $g = g + \nabla f_j(\tilde{\mathbf{x}}_0^s)$ locally,
5) **(End)**.
6) Compute the full gradient $\nabla f(\tilde{\mathbf{x}}_0^s) = \nabla f(\tilde{\mathbf{x}}_0^s) + \frac{1}{n} g$ globally and with **locks** .
7) **(Synchronization)** Wait for other threads to finish this step.

Part II: Variance Reduced Gradient Descent

1) **(Read)** Read the full gradient $\nabla f(\tilde{\mathbf{x}}_0^s)$ from the global memory to the local memory without locks, and set $k = 0$.
2) **(Loop)** while $k < m$
3) $k = k + 1$, globally,
4) **(Read)** $\tilde{\mathbf{x}}_k^s$ from the global memory with **locks** ,
5) **(Sample)** Randomly select a training samples i_k,
6) Compute $\mathbf{v}_k^s$ through Eq. (2) locally.
7) Update $\tilde{\mathbf{x}}_{k+1}^s = \tilde{\mathbf{x}}_k^s - \gamma \mathbf{v}_k^s$ globally and with **locks** .
8) **(End)**.
9) **(Synchronization)** Wait for other threads to this step and then set $\tilde{\mathbf{x}}_0^{s+1} = \tilde{\mathbf{x}}_m^s$ globally.

Algorithm 2 ASVRG

Input $\mathbf{x}_0^0$, epoch length m, step size γ, and $S = \lfloor K/m \rfloor$.
for $s = 0$ **to** $S - 1$ **do**
  $\mathbf{g}^s = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_0^s)$,
  for $k = 0$ **to** $m - 1$ **do**
    Randomly sample i_k from $1, 2, \cdots, n$,
    $\mathbf{v}_{\mathbf{j(k)}}^s = \nabla f_{i_k}(\mathbf{x}_{\mathbf{j(k)}}^s) - \nabla f_{i_k}(\mathbf{x}_0^s) + \mathbf{g}^s$,
    $\mathbf{x}_{k+1}^s = \mathbf{x}_k^s - \gamma \mathbf{v}_{\mathbf{j(k)}}^s$,
  end for k.
  $\mathbf{x}_0^{s+1} = \mathbf{x}_m^s$,
end for s.

The above algorithm is called ASVRG-atom, since the value of $\mathbf{x}$ is updated as an atom. Whenever $\mathbf{x}$ is being updated, it will be locked. So other workers cannot read or write it during the update. There are differences between ASVRG-atom and the serial SVRG during the "read" step, since when a thread has read $\mathbf{x}$ and is computing the gradient, other threads might update it during this time. So the gradient $f_{i_k}(\mathbf{x})$ might be computed from some early $\mathbf{x}$ instead of the current one during the asynchronous updates. We use $\mathbf{x}_{j(k)}^s$ to denote the early state of $\mathbf{x}$ which is used for calculating the gradient. Then the algorithm can be written as Algorithm 2. The advantage of ASVRG-atom is that the algorithm ensures the gradient to be calculated on a real state of $\mathbf{x}$. Namely, We have

$$\mathbf{x}_{j(k)}^s \in \{\mathbf{x}_1^s, \mathbf{x}_2^s, \cdots, \mathbf{x}_k^s\}. \tag{5}$$

ASVRG-wild

Now we describe ASVRG-wild, which is common in shared memory multi-core systems. We consider the case where the $\mathbf{x}$ is updated without locks, since the dimension of parameters is often large in real problems. In practice, we only need to change the steps (4) and (7) in Part II of ASVRG-atom to

4) read $\tilde{\mathbf{x}}$ from the global memory without locks.
7) update $\tilde{\mathbf{x}}_{k+1}^s = \tilde{\mathbf{x}}_k^s - \gamma \mathbf{v}_k^s$ without locks.

When $\mathbf{x}$ is updated without locks, it will cause inconsistent read at the "read" step. Unlike SVRG-atom, the value of $\mathbf{x}$ might not be a real state of $\mathbf{x}$ during the "read" step. Since if one thread reads the $\mathbf{x}$ when other thread is updating it, the thread will receive a "middle" state of $\mathbf{x}$. Some coordinates of $\mathbf{x}$ have been updated, while others have not. We still use $\mathbf{x}_{j(k)}^s$ to denote the state of $\mathbf{x}$ which is at the "read" time during the k-th iteration and is used for calculating the gradient. In this time, $\mathbf{x}_{j(k)}^s$ might not belong to $\{\mathbf{x}_1^s, \mathbf{x}_2^s, \cdots, \mathbf{x}_k^s\}$.

However, though $\mathbf{x}$ does not work as an atom, the update on a single coordinate can be considered to be atomic on GPU and Data Processing System (Niu et al. 2011), (Lian et al. 2015). To characterize this asynchronous implementation, (Lian et al. 2015) then define the update on each single coordinate of $\mathbf{x}$ and assume that the update order is random. They finally model the wild update as a Stochastic Coordinate Descent process. We do not follow their assumption.

We still define the update on the whole vector $\mathbf{x}$ and directly represent $\mathbf{x}^s_{j(k)}$. Since the update on a single coordinate is atomic, we have

$$\mathbf{x}^s_{j(k)} = \mathbf{x}^s_k - \sum_{l=1}^{k-1} \gamma I_{k(l)} \left(\mathbf{v}^s_{j(l)}\right), \tag{6}$$

where $I_{k(l)}$ is an $\mathcal{R}^d \to \mathcal{R}^d$ function, indicating whether the elements of $\mathbf{v}^s_{j(l)}$ have been returned from the local memory and written into $\mathbf{x}$ at the "read" step in the k-th iteration and d is the dimension of the variable $\mathbf{x}$. Suppose $\mathbf{v}^s_{j(l)}(p)$ is the p-th element of $\mathbf{v}^s_{j(l)}$ with p ranging from 1 to d. We have

$$I_{k(l)}(\mathbf{v}^s_{j(l)})(p) = \begin{cases} 0, & \text{if } \mathbf{v}^s_{j(l)}(p) \text{ has been returned,} \\ \mathbf{v}^s_{j(l)}(p), & \text{otherwise.} \end{cases} \tag{7}$$

Then in this way, ASVRG-wild can also be formulated as in Algorithm 2. One can find that Eq. (5) is actually a simply case of Eq. (6). So the difference between ASVRG-atom and ASVRG-wild is that $\mathbf{x}^s_{j(k)}$ satisfies both Eq. (5) and Eq. (6) in ASVRG-atom, while $\mathbf{x}^s_{j(k)}$ only satisfies Eq. (6) in ASVRG-wild. By using Eq. (6), we provide a unified convergence analysis for the two algorithm.

Convergence Analysis

In this section, we give a unified convergence analysis for ASVRG-atom and ASVRG-wild. It mainly consists of three parts. We first bound the variance of gradient in Lemma 1. Then we analyse the convergence rate in two cases. The first case ensures ASVRG to achieve linear speed up. In the second case, we analyse the convergence rate with no assumptions. The proofs can be found in Supplementary Material[2].

The most important value in the analysis of asynchronous algorithms is the delay parameter τ. We define that all the updates before the $(k-\tau-1)$-th iteration have been written into $\mathbf{x}$ at the "read" step of the k-th iteration. Thus Eq. (6) can be rewritten as:

$$\mathbf{x}^s_{j(k)} = \mathbf{x}^s_k - \sum_{l=k-\tau}^{k-1} \gamma I_{k(l)} \left(\mathbf{v}^s_{j(l)}\right). \tag{8}$$

τ actually indicates the number of processors that are involved in computation. From Algorithm 2, we have $1 \le \tau \le m$.

To prove convergence, we need to bound the variance of the gradient. In ASVRG, Eq. (4) changes to

$$\mathbb{E}\left(\|\mathbf{v}^s_{j(k)}\|^2\right) \le \mathbb{E}\left(\|\nabla f(\mathbf{x}^s_{j(k)}\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_{j(k)} - \mathbf{x}^s_0\|^2\right). \tag{9}$$

Then Lemma 1 builds the relation between $\mathbf{x}^s_{j(k)}$ and $\mathbf{x}^s_k$.

Lemma 1. *Suppose f_i ($i \in \{1, 2, \cdots, n\}$) have L-Lipschitz continuous gradients, and $\mathbf{x}$ is updated as in Algorithm 2. Then if the step size γ satisfies*

$$\gamma \le \min \frac{1}{L} \left\{ \frac{\rho_1 - 1}{2\sqrt{2}\rho_1\sqrt{\rho_2}}, \frac{\rho_2 - 1}{2\sqrt{2}\rho_1^{\frac{1}{2}}\rho_2^{\frac{3}{2}}\frac{\rho_1^{\frac{\tau}{2}}-1}{\sqrt{\rho_1}-1}} \right\}, \tag{10}$$

for some $\rho_1 > 1$ and $\rho_2 > 1$, then for any $s \ge 0$ and $k \ge 0$, we have

$$\begin{aligned} &\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_k)\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_k - \mathbf{x}^s_0\|^2\right) \\ \le\ &\rho_1\left[\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_{k+1})\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_{k+1} - \mathbf{x}^s_0\|^2\right)\right], \end{aligned} \tag{11}$$

and

$$\begin{aligned} &\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_{j(k)})\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_{j(k)} - \mathbf{x}^s_0\|^2\right) \\ \le\ &\rho_2\left[\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_k)\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_k - \mathbf{x}^s_0\|^2\right)\right]. \end{aligned} \tag{12}$$

For ASVRG-atom, ρ_2 can be set as ρ_1^τ if Eq. (11) is satisfied, since $\mathbf{x}^s_{j(k)}$ is some old value of $\mathbf{x}^s_k$. However, for ASVRG-wild, it is not true. The proof of Lemma 1 has two major distinctions. First, we analyse $\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_k)\|^2\right) + L^2\mathbb{E}\left(\|\mathbf{x}^s_k - \mathbf{x}^s_0\|^2\right)$, while the others (Liu et al. 2015), (Hsieh, Yu, and Dhillon 2015), (Peng et al. 2015) only consider $\mathbb{E}\left(\|\nabla f(\mathbf{x})\|^2\right)$ directly. Second, unlike (Lian et al. 2015), (Liu et al. 2015), (Hsieh, Yu, and Dhillon 2015), the update in our algorithm is defined on the whole vector $\mathbf{x}$ and our result does not depend on the dimension of the variable. This is because that we use Eq. (8) to represent $\mathbf{x}^s_{j(k)}$ and carefully bound $\mathbb{E}\left(\|\mathbf{x}^s_k - \mathbf{x}^s_{j(k)}\|^2\right)$ in the proof. Lemma 1 is the key result for analysing the convergence properties.

Now we demonstrate the convergence results. It uses the technique of the proof in serial SVRG (Reddi et al. 2016). The general results are shown in Theorems 1 and 2. We first consider the case when ASVRG can achieve linear speed up.

Theorem 1. *Suppose f_i ($i \in \{1, 2, \cdots, n\}$) have L-Lipschitz continuous gradients, and $\mathbf{x}$ is updated as in Algorithm 2. Assume that $\tau \le n^{\frac{\alpha}{2}}$ ($0 < \alpha \le 1$). Set $\gamma = \frac{\mu}{Ln^\alpha}$ with $0 < \mu \le \frac{1}{8(e-1)e}$ and $m = \lfloor n^{3\alpha/2} \rfloor$. Then we have*

$$\frac{1}{K}\sum_{s=0}^{S-1}\sum_{k=0}^{m-1}\mathbb{E}\left(\|\nabla f(\mathbf{x}^s_k)\|^2\right) \le \frac{n^\alpha(f(\mathbf{x}^0_0) - f(\mathbf{x}^*))}{K\nu}, \tag{13}$$

where $K = mS$, $\nu = \frac{1}{3}\mu$, and $f(\mathbf{x}^*)$ is the minimal value of $f(\mathbf{x})$.

We rewrite the above results in terms of IFO calls in the following corollary. The IFO calls have included the n IFO calls to compute the full gradient for every m iterations.

Corollary 1. *Suppose f_i ($i \in \{1, 2, \cdots, n\}$) have L-Lipschitz continuous gradients and $\tau \le n^{\frac{\alpha}{2}}$ ($0 < \alpha \le 1$). With the parameters in Theorem 1, the IFO complexity of Algorithm 2 for achieving an ϵ-accurate solution is:*

$$\textit{IFO calls} = O\left(n^{\max\left(\alpha, 1-\frac{\alpha}{2}\right)}/\epsilon\right). \tag{14}$$

Corollary 1 demonstrates the interplay between the step size and the IFO complexity. The result is similar to the serial SVRG (Reddi et al. 2016). When $\tau \le n^{1/3}$, the number of IFO calls is minimized when $\alpha = 2/3$ and it is $O\left(\frac{n^{2/3}}{\epsilon}\right)$. This shows that ASVRG can achieve linear speed up (to the stationary point) when $\tau \le n^{1/3}$.

Now we demonstrate the result with no assumption on τ.

[2]Supplementary Material can be downloaded from: http://www.cis.pku.edu.cn/faculty/vision/zlin/zlin.htm

Theorem 2. *Suppose f_i $(i \in \{1, 2, \cdots, n\})$ have L-Lipschitz continuous gradients, and* $\mathbf{x}$ *is updated as in Algorithm 2. Set $\gamma = \mu/(Ln^{\alpha}\tau^{\beta})$ ($0 < \mu \leq \frac{1}{8(e-1)e}$, $0 < \alpha \leq 1$, and $0 < \beta \leq 1$) and $m = \lfloor n^{\frac{3\alpha}{2}}\tau^{\frac{3\beta-1}{2}} \rfloor$. Then we have*

$$\frac{1}{K}\sum_{s=0}^{S-1}\sum_{k=0}^{m-1}\mathbb{E}\left(\|\nabla f(\mathbf{x}_k^s)\|^2\right) \leq \frac{n^{\alpha}\tau^{\beta}(f(\mathbf{x}_0^0) - f(\mathbf{x}^*))}{K\nu}, \quad (15)$$

where $K = mS$ and $\nu = \frac{1}{3}\mu$.

Corollary 2. *Suppose f_i $(i \in \{1, 2, \cdots, n\})$ have L-Lipschitz continuous gradients. With the parameters in Theorem 2, the IFO complexity of Algorithm 2 for achieving an ϵ-accurate solution is:*

$$\textit{IFO calls} = O\left(n^{\max\left(\alpha, 1-\frac{\alpha}{2}\right)}\tau^{\max\left(\beta, \frac{1-\beta}{2}\right)}/\epsilon\right). \quad (16)$$

From Corollary 2, the number of IFO calls is minimized when $\alpha = \frac{2}{3}$ and $\beta = \frac{1}{3}$, which is $O\left(\frac{n^{2/3}\tau^{1/3}}{\epsilon}\right)$. Since $\tau \leq m = n$, $n^{2/3}\tau^{1/3} \leq n$. This shows that with asynchronous updating, SVRG still has a less number of IFOs when compared with GD.

The following theorem gives a probability estimate on the convergence of Algorithm 2.

Theorem 3. *Suppose f_i $(i \in \{1, 2, \cdots, n\})$ have L-Lipschitz continuous gradients, and* $\mathbf{x}$ *is updated as in Algorithm 2. Then for $\epsilon > 0$ and $\eta \in (0, 1)$ and $K = Sm$, we have the probability*

$$\mathcal{P}\left(\frac{1}{K}\sum_{s=0}^{S-1}\sum_{k=0}^{m-1}\|\nabla F(\mathbf{x}_k^s)\|^2 \leq \epsilon\right) \geq 1 - \eta, \quad (17)$$

provided that one of the following conditions holds: when $\tau \leq n^{\alpha/2}$, we require

$$K \geq \frac{n^{\alpha}\left(F(\mathbf{x}^0) - F(\mathbf{x}^*)\right)}{\nu\epsilon\eta}, \quad (18)$$

and the parameters are chosen as in Theorem 1, while there is no assumption on τ, we need

$$K \geq \frac{n^{\alpha}\tau^{\beta}\left(F(\mathbf{x}^0) - F(\mathbf{x}^*)\right)}{\nu\epsilon\eta}, \quad (19)$$

and the parameters are chosen as in Theorem 2.

Distinguished from the analysis of ASGD (Lian et al. 2015), we do not assume that the update order on coordinates is random in the "wild" scheme and give unified proofs for the two schemes. For ASGD, the convergence rate is only an asymptotic $O\left(\frac{1}{\epsilon^2}\right)$. We show that with the VR trick, ASGD can be accelerated to a non-asymptotic $O(\frac{n^{\frac{2}{3}}}{\epsilon})$ convergence rate when $\tau \leq n^{\frac{1}{3}}$ and a non-asymptotic $O(\frac{n^{\frac{2}{3}}\tau^{\frac{1}{3}}}{\epsilon})$ convergence rate where there is no assumption on τ. When compared with the serial SVRG, ASVRG can achieve linear speed up when $\tau \leq n^{\frac{1}{3}}$ and also has less number of IFOs than GD when there is no assumption on τ.

Experiments

In this section, we conduct experiments on a shared memory multi-core system to validate the efficiency of our algorithm empirically. We directly show the experimental results of ASVRG-wild as it is faster and more suitable for the shared memory multi-core system. We also test the speedup property of ASVRG-atom, which is shown in Supplementary Material. Due to the locks, it is slower than ASVRG-wild. Our experiments consist of two parts. The first part aims to validate the speedup property. In the second part, we do a similar experiment to that in (Lian et al. 2015) to compare our algorithm with ASGD to show the superiority of our algorithm in speed. Since the advantages in speed for asynchronous algorithms over synchronous algorithms have been widely witnessed in many literatures (Hsieh, Yu, and Dhillon 2015), (Agarwal and Duchi 2011), (Niu et al. 2011), we ignore the experiment of comparing our algorithm with synchronous SVRG. Due to much more locks, synchronous SVRG is slower than ASVRG-atom. For a fair comparison, we implement all methods in C++ using POSIX threads as the parallel programming framework. All the experiments are performed on an Intel multi-core 4-socket machine with 128 GB memory. Each socket is associated with 8 computation cores. A variant that we adopt in experiments is that we implement all the algorithms in a mini-batch mode, which is a common implementation in neural networks. The convergence analysis for ASVRG can be extended to this mode.

Following (Lian et al. 2015), we focus on two types of speedup: iteration speedup and running time speedup. The iteration speedup is exactly the speedup we discussed in the whole paper. Given T workers, it is computed as

$$\text{iteration speedup} = \frac{\text{\# of itertions using one worker}}{\text{\# of iterations using } T \text{ workers}} \times T,$$

where # is the iteration count when the same level of precision is achieved. This speedup is less affected by the hardware. The running time speedup is the actual speedup. It is defined as:

$$\text{running time speedup} = \frac{\text{total running time using one worker}}{\text{total running time using } T \text{ workers}}.$$

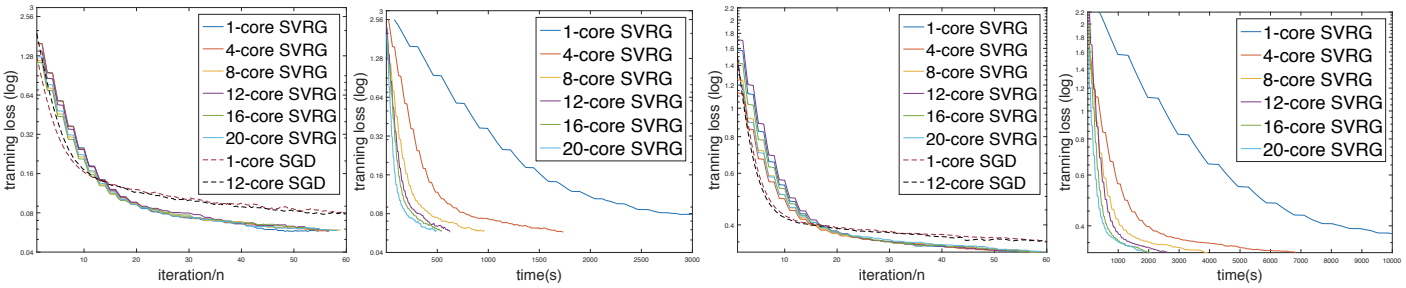
The running time speedup is seriously affected by the hardware. It is less objective than the iteration speedup.

Speedup Experiment

We experiment on the problem of multiclass classification using neural networks. It is a typical nonconvex problem in machine learning.

Experimental Setup. Following (Reddi et al. 2016), we train a neural network with one fully connected layer of 100 nodes. We experiment on two dataset: MNIST dataset[3] and CIFAR10 dataset (Krizhevsky and Hinton 2009). Both the two datasets have ten classes. They are widely used for testing neural networks. More details about the datasets can be found in Table 2. The data are normalized to the interval $[0, 1]$ before the experiment. An additional experiment in

[3] http://yann.lecun.com/exdb/mnist/



(a) Loss vs. iteration on MNIST (b) Loss vs. time on MNIST (c) Loss vs. iteration on CIFAR10 (d) Loss vs. time on CIFAR10

Figure 1: Results of the speedup experiment. For curves of loss against iterations, the horization axis is the number of effective pass through the data, which has included the cost of calculating full gradients for SVRG.

Table 2: More details about MNIST and CIFAR10.

Datasets	Type	# Images	# Params
MNIST	28×28 grayscale	$60K$	$79.5K$
CIFAR10	32×32 RGB	$50K$	$308.3K$

Table 3: Iteration and running time speedup over SVRG on MNIST and CIFAR10. (Thr- and Iter, are short for thread and iteration, respectively.).

		thr-1	thr-4	thr-8	thr-12	thr-16	thr-20
Mnist	iter.	1	3.94	7.55	11.85	15.53	19.28
	time	1	3.59	6.47	9.97	11.44	12.58
Cifar	iter.	1	4.01	7.92	12.15	15.59	19.31
	time	1	3.96	6.87	10.31	13.02	14.53

which we train a neural network with 7 layers on MNIST is shown in Supplementary Material.

Parameters and Initialization. For SVRG, we choose a fixed step size, and choose γ that gives the best performance on one core. When there are more than one core, the step size does not change. For SGD, the step size is chosen based on (Reddi et al. 2016), which is $\gamma_t = \gamma_0(1+\gamma' \lfloor t/n \rfloor)^{-1}$, where γ_0 and γ' are chosen to give the best performance. We use the normalized initialization in (Glorot and Bengio 2010), (Reddi et al. 2016). The parameters are chosen uniformly from $[-\sqrt{6/(n_i+n_o)}, \sqrt{6/(n_i+n_o)}]$, where n_i and n_o are the numbers of input and output layers of the neural networks, respectively. We choose a mini-batch size to be 100, which is a common setting in training neural networks.

Results. We draw the curves of objective loss against iterations and running time in Figure 1, and report their speedup in Table 3. From the results, we obtain the following conclusions. First, the linear speedup is achievable through iteration speedup. Second, due to the hardware, time speedup is lower than iteration speedup. Third, ASVRG still has an obvious actual (time) speedup when compared with serial SVRG, e.g., there are 12 times speedup on 20 cores.

Efficiency Validation

To demonstrate the efficiency of our ASVRG, we do a similar experiment to that in (Lian et al. 2015) to compare with ASGD. Following (Lian et al. 2015), we generate the synthetic data from a fully connected neural network with 5 layers ($400 \times 100 \times 50 \times 20 \times 10$) and $46,380$ parameters totally. The input vector and all parameters are generated from $\mathcal{N}(0,1)$ Gaussian distribution. The output vector is constructed by applying the network parameter to the input vector plus some Gaussian random noise. We generate $40,000$ samples.

Like (Lian et al. 2015), we focus on ℓ_2 norm of the gradients. The parameters in the two algorithms are tuned on 12 cores to give the best results. For ASGD, we choose the mini-batch size to be 50, and the step size to be 10^{-4}, which we find is better than the setting used in (Lian et al. 2015).

Figure 2 draws the curves of $\|\nabla f(\mathbf{x})\|^2$ against running time using 8, 12, 20 cores, respectively. Like serial SVRG, ASVRG is not faster at the early stage when compared with ASGD. But after dozens of epochs, the norm of gradient by ASVRG decreases faster. This demonstrates that ASVRG has a faster convergence rate than ASGD does.

Conclusion

This paper proposes an asynchronous variant of SVRG on nonconvex problems. We give the condition on the delay parameter τ to make the asynchronous algorithm achieve linear speed up. We also analyse the convergence rate with no assumption on τ. We experiment on a shared memory multicore system to demonstrate the efficiency of the proposed ASVRG algorithm.

Acknowledgements

Zhouchen Lin is supported by National Basic Research Program of China (973 Program) (grant no. 2015CB352502), National Natural Science Foundation (NSF) of China (grant nos. 61625301 and 61231002), and Qualcomm.

References

Agarwal, A., and Duchi, J. C. 2011. Distributed delayed stochastic optimization. In *Proc. Conf. Advances in Neural Information Processing Systems*.

Allen-Zhu, Z., and Hazan, E. 2016. Variance reduction for faster non-convex optimization. In *Proc. Int'l. Conf. on Machine Learning*.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al.

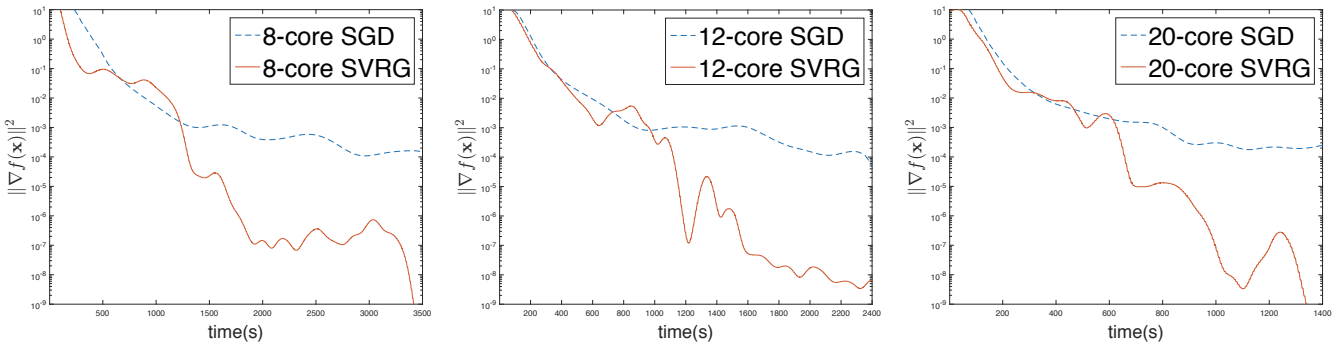


Figure 2: Experimental results of efficiency validation. The figures from left to right represent the results on 8, 12, and 20 cores, respectively.

2012. Large scale distributed deep networks. In *Proc. Conf. Advances in Neural Information Processing Systems*.

Defazio, A.; Bach, F.; and Lacoste-Julien, S. 2014. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Proc. Conf. Advances in Neural Information Processing Systems*.

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

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. Int'l. Conf. on Artificaial Intelligence and Statistics*.

Hsieh, C.-J.; Yu, H.-F.; and Dhillon, I. S. 2015. PASSCoDe: Parallel asynchronous stochastic dual co-ordinate descent. In *Proc. Int'l. Conf. on Machine Learning*.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Proc. Conf. Advances in Neural Information Processing Systems*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Technical Report* 1(4):7.

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*.

Liu, J.; Wright, S. J.; Ré, C.; Bittorf, V.; and Sridhar, S. 2015. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research* 16(285-322):1–5.

Nesterov, Y. 2013. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

Niu, F.; Recht, B.; Re, C.; and Wright, S. 2011. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *Proc. Conf. Advances in Neural Information Processing Systems*.

Paine, T.; Jin, H.; Yang, J.; Lin, Z.; and Huang, T. 2013. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*.

Peng, Z.; Xu, Y.; Yan, M.; and Yin, W. 2015. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *arXiv preprint arXiv:1506.02396*.

Petroni, F., and Querzoni, L. 2014. Gasgd: stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning. In *Proc. ACM Conf. on Recommender Systems*, 241–248. ACM.

Reddi, S. J.; Hefny, A.; Sra, S.; Póczós, B.; and Smola, A. 2015. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Proc. Conf. Advances in Neural Information Processing Systems*.

Reddi, S. J.; Hefny, A.; Sra, S.; Póczós, B.; and Smola, A. 2016. Stochastic variance reduction for nonconvex optimization. In *Proc. Int'l. Conf. on Machine Learning*.

Schmidt, M.; Roux, N. L.; and Bach, F. 2013. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*.

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

Yun, H.; Yu, H.-F.; Hsieh, C.-J.; Vishwanathan, S.; and Dhillon, I. 2014. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *Proc. of VLDB Endowment* 7(11):975–986.

Zhang, R., and Kwok, J. T. 2014. Asynchronous distributed ADMM for consensus optimization. In *Proc. Int'l. Conf. on Machine Learning*.